Structures et systèmes répartis

Gilles Trédan

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Gilles TREDAN

préparée à l'unité de recherche 6073: IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
Maths-STIC

Structures et systèmes répartis

Thèse soutenue à Rennes
le 26 Novembre 2009

devant le jury composé de :

Carole DELPORTE-GALLET
Professeur à l'Université Paris 7/rapporteur

Sébastien TIXEUIL
Professeur à l'Université Paris 6/rapporteur

Olivier RIDOUX
Professeur à l'Université de Rennes 1/examinateur

Olivier BEAUMONT
Directeur de Recherche au LaBRI/examinateur

Michel RAYNAL
Professeur à l'Université de Rennes 1/examinateur

Achour MOSTEFAOUI
Maître de Conférences habilité à l'Université de Rennes 1/directeur de thèse

Table des matières

Liste des figures **III**

1 Pourquoi les systèmes répartis **1**
1.1 Un système réparti . . . 2
1.2 Décomposer les problèmes . . . 2
1.3 Capturer la difficulté . . . 3

2 Modèles dans les systèmes répartis **5**
2.1 Familles de modèles . . . 5
2.1.1 Modèles des interactions . . . 6
Acteurs des interactions . . . 6
Déroulement des interactions . . . 6
2.1.2 Modèles des entités . . . 7
Entités faillibles . . . 7
Byzantins . . . 8
Connaissance des processus . . . 8
Messages . . . 9
2.2 Couches et boites noires . . . 9
2.2.1 Hiérarchie de modèles . . . 9
2.3 Enjeux de la conception en couches . . . 9
2.3.1 Complexité . . . 10
2.3.2 Fiabilité . . . 11
2.3.3 Structure . . . 11

3 Étude des structures naturelles des systèmes répartis **13**
3.1 Estimation du nombre de processus vivants dans un système asynchrone . . . 14
3.1.1 Motivation . . . 14
Incertitude des systèmes asynchrones . . . 14
3.1.2 Travaux connexes . . . 15
3.1.3 Modèle . . . 15
3.1.4 Estimation du nombre de processus vivants . . . 16
Protocole basé sur les horloges globales . . . 16
Protocole base sur les horloges locales . . . 19
Lorsque n est inconnu . . . 21
3.1.5 Évaluation expérimentale . . . 21
Le modèle de simulation . . . 21
Quelle est la précision du protocole ? . . . 23
Quelle est la vitesse de convergence du protocole ? . . . 24
3.1.6 Impact du réseau sous-jacent . . . 25

3.1.7 Compenser les effets néfastes de la couche sous-jacente 26
3.2 Centralité du second ordre : calcul réparti de l'importance des noeuds 28
3.2.1 Intérêt des mesures de centralité 28
3.2.2 Modèle et principes du protocole 29
3.2.3 Travaux connexes 30
Niveau "Macro" : caractérisation globale des graphes 30
Niveau "micro" : influence des individus dans le graphe 31
3.2.4 Centralité du second ordre 32
Intuition 32
L'algorithme 33
3.2.5 Modélisation des temps de retour 34
Analyse théorique 34
Résultat pour 3 classes de graphes 38
Application à des graphes spécifiques : signatures 40
3.2.6 Expérimentation 41
Degré des noeuds et débiaisement 42
Vitesse de convergence 43
Identification des goulots d'étranglement 43
Vers une détection répartie 44
3.3 Conclusion 46

4 Exploiter les structures **49**
4.1 Consensus byzantin dans un réseau très asynchrone 50
4.1.1 Motivation 50
4.1.2 Travaux connexes 51
4.1.3 Modèle de calcul et définition du problème 52
Modèle 52
Le consensus 54
4.1.4 Le protocole 54
4.1.5 Correction du protocole 56
4.1.6 Remarques 58
Minimalité 58
Structure du réseau 59
4.2 Établissement de secret dans les réseaux de capteurs 59
4.2.1 Motivation 60
4.2.2 Travaux connexes 60
4.2.3 Modèle 61
Communications 61
Rapport au temps 62
Adversité 62
problème 62
4.2.4 Description de l'algorithme 63
Etablissement de clés secrètes à un saut 63
De la sécurité du voisinage à la sécurité à plusieurs sauts 65
4.2.5 Simulations 67
Point de vue local 67
Point de vue global 69
Gain de confidentialité 69
4.2.6 Structures régulières 71

Caractérisation des structures favorables à l'établissement des liens sécurisés 71
4.2.7 Comportement byzantin 73
4.3 Conclusion 74

5 Expliciter les structures **77**
5.1 Coordonnées virtuelles pour les réseaux de capteurs 78
5.1.1 Travaux connexes 79
5.1.2 Principe des coordonnées 80
5.1.3 Définition de la bordure du système 81
La notion de bordure 81
Propriétés du détecteur de bordure 82
Détection de bordure dans des ensembles convexes 82
5.1.4 Modèle 83
5.1.5 Segmentation de la bordure 84
5.1.6 Définition des coordonnées 85
5.1.7 Analyse de la complexité 86
5.2 Des coordonnées virtuelles à la structuration géographique 87
5.3 Evaluation des performances 90
5.3.1 Coûts de communication 90
5.3.2 Précision 91
5.3.3 Routage et densité 92
5.3.4 Routage et pertes 93
5.3.5 Taille des parties 95
5.4 Conclusion 96

6 Conclusion **99**
6.1 Contexte 99
6.2 Résumé des chapitres 100
6.3 Perspectives et défis 101

Index **102**

Bibliographie **105**

Table des figures

3.1 L'opération estimate() (version basée sur des horloges globales) 17
3.2 L'opération estimate() (version basée sur des horloges locales non synchronisées) 19
3.3 Utilisation de la relation *happened before* 20
3.4 Modélisation du réseau sous-jacent 22
3.5 Probabilité de ne pas suspecter de processus correct 23
3.6 Résultat de simulation : A quelle vitesse le protocole procure-t-il une estimation précise ? (nombre de rondes nécessaires au rétablissement en fonction du nombre de fausses suspicions initiales) 23
3.7 Résultat de simulation : A quelle vitesse le protocole procure-t-il une estimation précise ? (probabilité de rétablissement en fonction du temps) 24
3.8 Temps de convergence pour différents délais inter-routeurs 26
3.9 Temps d'arrivée des messages 26
3.10 Temps de convergence pour différents délais inter-routeurs — avec délai d'émission 27
3.11 Illustration des faiblesses de la centralité d'intermédiarité : c obtient un score bas malgré son importance 32
3.12 Exemple d'un graphe Barbell 33
3.13 Calcul de la centralité du second ordre 35
3.14 Histogramme de la valeur théorique de la centralité du second ordre pour 4 graphes particuliers 41
3.15 Centralité du second ordre : marche débiaisée contre marche aléatoire classique . 42
3.16 L'algorithme converge vers les valeurs théoriques, pour 4 classes de graphes . . . 43
3.17 Evolution de la position des ponts parmi les plus basses centralités du réseau . . . 44
3.18 (a) Répartition de 10^3 noeuds sur une topologie 2-D comportant un goulot d'étranglement, (b) Distribution des écarts types, après 1×10^6 pas 45
3.19 Comparaison noeud à noeud des temps de passage pour un graphe clusterisé . . . 46

4.1 Protocole de consensus byzantin (suppose une $\diamond 2t$-bisource) 56
4.2 Exemple d'établissement de secret à un saut 63
4.3 Établissement des clés à un saut 64
4.4 Exemple d'utilisation des chemins disjoints : une partie de la clé transite par C, l'autre partie transite par E, D et F 66
4.5 Exemple de chemin $\mathcal{C}p_{ij}^2$ 66
4.6 Exemple de transmission multisaut 67
4.7 Impact de la densité sur le nombre de liens du système 68
4.8 Impact de la densité sur le nombre de noeuds partageant des clés 69
4.9 Impact du protocole sur le routage glouton et la confidentialité 70
4.10 Structures particulières 72
4.11 Illustration de l'impact de l'angle des routes sur la zone de vulnérabilité : plus l'angle est petit plus la zone vulnérable est grande. 74

4.12 Probabilité d'interception d'un chemin p^+ par un noeud byzantin muet 75

5.1 Vincos en termes de coût, précision et connaissance externe. 79
5.2 Un exemple de coordonnées virtuelles. 81
5.3 Chaque sonde transporte le voisinage du deuxième noeud la relayant. 84
5.4 Les 4 segments définis dans un réseau de 2000 noeuds. 85
5.5 Algorithme de la sonde (Probe) 86
5.6 (a)-(c) Exemples de structuration géométrique. 88
5.7 (a)-(c) Partitionnements Fonctionnels : (a) Partitionnement en treillis. (b) Bandes verticales (c) Œil. 89
5.8 Analyse du coût en fonction de la taille du réseau, pour $d = 4$, portée radio de 40-unités. 90
5.9 Coût de Vincos, pour $d = 8$, en fonction de la densité du système. 91
5.10 (a) Ratio des simulations ayant conduit à des systèmes précis. (b) Histogramme des distances entre noeuds homonymes dans un système non précis de 1000 noeuds. 92
5.11 Impact de la densité sur le taux de succès du routage. 93
5.12 (a) Réseau circulaire plein (b) Réseau circulaire - trou circulaire (c) Réseau circulaire - trou irrégulier. 94
5.13 Taux de perte moyens en fonction de la distance émetteur-récepteur et de r. . . . 94
5.14 Taux d'échec du routage en fonction des pertes et de la forme du réseau. 95
5.15 (a) Qualité du partitionnement géographique (b) Qualité du partitionnement fonctionnel. 96

Chapitre 1

Pourquoi les systèmes répartis

Il est raisonnable de penser que les premiers concepteurs d'ordinateurs avaient une vision assez pragmatique de leurs recherches. Souvent, les premiers ordinateurs sont ainsi présentés comme les descendants des calculatrices, permettant d'effectuer plus de calculs, plus rapidement. Rien ne laissait alors présager l'importance que prendrait l'informatique dans le monde. Dans la logique de la vision de l'ordinateur comme une calculatrice, seuls quelques ordinateurs étaient nécessaires, principalement utilisés par des universitaires ou des militaires. Déjà cependant, l'intérêt de l'informatique se dessine : remplacer l'homme de façon sûre dans l'accomplissement de tâches répétitives.

La suite est connue : les progrès de l'électronique permettent une miniaturisation de l'équipement informatique, ainsi qu'une accessibilité accrue. L'informatique se voit ainsi attribuer plus de tâches. Le côté prédictible de l'informatique et sa relative fiabilité permettent de lui confier maintenant des tâches importantes. L'imaginaire public commence à prendre conscience du pouvoir de l'outil informatique, et l'idée d'un monde massivement informatisé émerge. Á l'époque, l'idée dominante est celle d'un grand ordinateur central, plutôt humanisé, connaissant tout et contrôlant tout.

Comme l'illustrent bien les récits de science fiction, confier l'intégralité (ou ne serait-ce qu'une partie) du monde à un unique ordinateur est une solution à double tranchant. D'un côté, elle promet une efficacité sans égal : l'unique ordinateur ayant accès à un maximum possible de connaissances, il est capable de faire les meilleurs choix possibles. De plus, étant la seule entité à prendre des décisions, celles-ci sont forcément cohérentes. De l'autre côté, le danger d'une telle centralisation est capturé par l'expression « mettre ses oeufs dans le même panier ». En effet, le monde informatisé dépend intégralement du bon fonctionnement de cet unique ordinateur. Si cet unique ordinateur venait à faillir, les conséquences seraient dramatiques (la grosse omelette).

Est-il possible de faire un ordinateur parfait ? A priori, non. D'une part, l'ordinateur est un produit complexe, fait de multiples composants faillibles, devant opérer dans des conditions très précises : un hypothétique ordinateur parfait ne saurait fonctionner hors d'un milieu lui aussi parfait . . . D'autre part, rappelons que l'ordinateur est un produit fortement soumis aux défaillances humaines, qui sont difficilement suppressibles. Ainsi, on estime que 70% des incidents survenus dans des centrales nucléaires aux États Unis sont engendrés par des défaillances humaines [Joh03].

C'est donc l'impasse. D'un côté l'indépendance efficace et fiable d'un ordinateur fonctionnant bien nous donne envie de lui confier énormément de tâches importantes. De l'autre, confier d'énormes responsabilités à une entité que l'on sait imparfaite n'est pas une bonne idée. A qui allons-nous pouvoir confier ces tâches importantes ? Où allons-nous mettre nos oeufs ?

Une idée vient assez naturellement : mettre plus de paniers.

1.1 Un système réparti

Un *système réparti* est constitué par l'interaction d'entités autonomes. *Entités autonomes* désigne traditionnellement tout circuit disposant d'une capacité de calcul. Il est assez difficile de préciser cette définition de l'entité sans réduire les champs d'application du système réparti. L'ordinateur de bureau est l'entité la plus fréquemment rencontrée, mais la miniaturisation ouvre chaque jour de nouveaux champs. Ainsi les téléphones portables, les netbooks, ou encore les capteurs intelligents constituent d'autres exemples d'entités. Par la suite, nous parlerons de noeud, ou de processus. L'*interaction*, quant à elle, impose un moyen et une volonté de communication. Les moyens de communiquer sont eux aussi vastes, et nous verrons qu'ils impactent fortement la forme et les capacités du système réparti. Là encore, il est difficile d'être plus précis sans exclure une partie des systèmes répartis. Le paradigme du moyen de communication est le réseau filaire Ethernet, éventuellement étendu par internet. Mais citons aussi les réseaux sans fils, comme le *Bluetooth* ou le *Wifi*, qui au contraire du réseau filaire ne permettent que rarement la communication permanente et continue de toutes les entités du réseau. Enfin, aux antipodes des réseaux, citons les mémoires partagées qui permettent aux entités de communiquer en lisant et écrivant une mémoire partagée accessible par toutes les entités.

L'image de la fourmilière permet d'illustrer les systèmes répartis : chaque ordinateur est une fourmi, et l'ensemble des ordinateurs collaborant est la fourmilière. Chaque fourmi, prise seule, semble avoir un comportement simple et rudimentaire. Mais d'un point de vue global, la fourmilière a un comportement cohérent et efficace. Le plus impressionnant est d'observer la robustesse d'une telle structure : si l'on écrase la moitié des fourmis composant la fourmilière, elle continuera à fonctionner.

Les systèmes répartis semblent être la solution idéale pour obtenir la robustesse et la continuité de service tant espérée. Cette solution a cependant un inconvénient majeur : sa complexité. Pour filer la métaphore de la fourmilière, le concepteur du système réparti veut concevoir la fourmilière de façon à ce qu'elle remplisse au mieux la tâche de son souhait. Cependant, il doit pour cela spécifier le comportement de chaque fourmi. Ainsi, il va devoir définir le comportement de chaque fourmi de façon à ce que la somme des actions et des interactions des fourmis de la fourmilière produise le résultat demandé. Et cela n'est pas facile.

Il est possible d'imputer la complexité de conception à deux phénomènes. Le premier est l'explosion des cas : dès qu'une « fourmi » devient un peu complexe, par exemple en possédant plusieurs états distincts, les interactions qu'elle peut avoir avec d'autres fourmis du même type sont excessivement complexes, et considérer tous les cas (p.ex. tous les k-uplets d'états possibles) devient fastidieux. Le second phénomène provient des effets de bord. De la même manière qu'il est difficile de modeler une société en contraignant les comportements des individus qui la composent, il est difficile d'intuiter l'influence à large échelle du changement d'un paramètre local, notamment en raison des multiples interactions évoquées plus haut. Chaque choix dans la programmation du comportement de la fourmi produit de multiples effets sur l'ensemble de la fourmilière qu'il est difficile d'intuiter, d'observer et même parfois, de comprendre.

Nous voilà donc face à la nécessité d'étudier et de comprendre les systèmes répartis. C'est l'objectif des travaux présentés dans la suite de ce document. Le parti pris de ce document est d'insister sur les similarités entre l'étude du réparti et l'étude de n'importe quel autre système, par exemple physique, biologique ou social.

1.2 Décomposer les problèmes

En informatique, comme dans tout domaine de l'ingénierie en général, l'objectif est de faire simple en découpant des problèmes compliqués en de multiples sous tâches. C'est l'application du

second principe de Descartes (dans Discours de la méthode): *"(. . .) diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre."* Chaque sous tâche, prise indépendamment, se doit d'être simple. C'est-à-dire qu'elle repose sur des hypothèses clairement identifiées et remplit des services clairement définis.

Il est possible de faire un parallèle avec la cuisine. Une recette définit une réalisation compliquée du point de vue de la liste des actions effectuées par le cuisinier sur son environnement (la cuisine, les ingrédients). Afin de faciliter la compréhension, la conception et la mémorisation de telles recettes, des *primitives* ont été définies. Ce sont ces primitives qui font les connaissances de base du cuisinier (saisir, déglacer, émincer). Certaines recettes supposent même que le cuisinier dispose de primitives encore plus puissantes (réaliser un roux, faire un caramel).

Cet éclatement a plusieurs avantages :

- La recette devient compacte. En effet, si la recette devait être exprimée sans ces primitives, c'est-à-dire en une suite d'actions compréhensibles par une personne n'ayant jamais vu de cuisine, sa représentation serait lourde et difficilement manipulable.
- Cet éclatement des actions permet aussi un diagnostic plus efficace. Ainsi un cuisinier ratant toutes ses viandes sait qu'il doit sûrement réviser sa primitive saisir. Il lui est d'ailleurs possible de trouver toutes les informations nécessaires à la bonne réalisation de cette technique, dans des livres ou en discutant avec d'autres cuisiniers ; en effet saisir est une primitive communément employée et de ce fait une technique bien étudiée et documentée.
- Les recettes deviennent plus génériques. Ainsi en spécifiant « saisir », la recette laisse le cuisinier libre de la façon dont il va saisir sa viande. Ceci permet par exemple à l'étudiant possédant des plaques électriques, pourvu qu'il trouve une façon satisfaisante de saisir, d'utiliser le même livre de recettes que le cuisinier professionnel sur son piano à gaz. On dira que l'implémentation (la suite d'actions effectuées pour saisir) est laissée libre. Ceci confère un caractère générique à la recette.

En informatique, de telles décompositions sont fréquentes. La conception objet traduit par exemple bien cette volonté : transformer le compliqué en complexe. Ainsi, chaque « objet » est un élément simple, assurant une fonction bien définie à l'aide de moyens clairement établis. La complexité du système vient de l'assemblage évolué de tels objets.

1.3 Capturer la difficulté

Les systèmes répartis, comme décrits plus haut, forment un vaste univers de systèmes. Afin de pouvoir travailler correctement, il faut borner les études et les solutions proposées à des sous parties de cet univers. C'est le rôle du *modèle* : il va « sélectionner » la sous partie de l'univers qui nous intéresse. Ce « filtrage » a un intérêt : il donne des prises sur les systèmes : cela revient à définir un ensemble de systèmes possédant des points communs. Il devient alors possible d'utiliser ces points communs pour travailler.

Dans le cas de systèmes réels, il faut aussi prendre conscience de la vaste étendue de paramètres influant sur l'objet d'étude. Ces systèmes ont pour objectif d'être déployés à l'échelle de la planète, avec pour conséquence une exposition à des phénomènes très divers qu'il faut pouvoir représenter en laboratoire. Un des défis consiste à capturer l'essentiel de ces phénomènes (c'est-à-dire la partie qui influence notre système) d'une façon suffisamment synthétique pour la manipuler. Un composant électronique, par exemple, se trouve dans un environnement physique que l'on imagine bien décrit par une combinaison de paramètres (p.ex. $\{pression, temperature, hygrometrie\}$). Décrire la topologie sur laquelle est déployée un réseau de capteurs est beaucoup moins aisé.

Chapitre 2

Modèles dans les systèmes répartis

Dans ce chapitre, nous allons explorer les modèles classiques des systèmes répartis. Nous verrons aussi qu'il existe en informatique théorique de grandes familles de modèles qui sont déclinées de multiples manières.

Nous avons, dans le chapitre précédent, définit les modèles comme une interface entre la théorie et le réel. On leur attribue traditionnellement 2 fonctions (voir *e.g.* [CDK05]) :

- Abstraire les systèmes réels et les regrouper. Chaque système réel est unique et proposer une solution pour chacun d'eux n'a pas de sens. Il faut les regrouper selon les points communs qu'ils exhibent et qui sont utiles au problème.
- Proposer un ensemble de propriétés aux théoriciens. Un modèle est une série d'hypothèses décrivant les systèmes manipulés. De ces hypothèses découle un certain nombre de propriétés. Ainsi il est prouvé que certaines opérations sont possibles ou au contraire impossibles dans certains modèles. Travailler sur un modèle permet d'utiliser toute la trousse à outils développée pour ce modèle par la communauté.

Le modèle constitue ainsi un pont et un point de passage obligé entre le théorique et le réel.

On peut dire d'un modèle qu'il est bon lorsqu'il est :

Sensé le modèle décrit effectivement bien la réalité dans des cas courants

Puissant les hypothèses fournies par le modèle sont utiles et permettent des démonstrations efficaces

Simple il est facile de comprendre les phénomènes décrits dans le modèle utilisé. Il est aussi facile d'intuiter et de raisonner en utilisant ce modèle.

Notons cependant qu'il n'existe aucune garantie sur le nombre de systèmes réels qu'un modèle capture, si bien qu'il existe des modèles « théoriques » qui ne capturent aucun système physique. Ces modèles ne sont cependant pas dénués d'intérêt puisqu'ils permettent de placer des repères dans le champ des possibles (voir la notion de réalisme de [DGFG02]).

Il est important de souligner que les modèles employés ici sont adaptés à notre objectif qui est celui de créer des algorithmes pour les systèmes répartis. Un électronicien cherchant à optimiser la durée de vie d'un réseau de capteurs utilisera d'autres modèles, adaptés à son problème.

2.1 Familles de modèles

Nous avons défini un système réparti comme l'interaction d'entités autonomes. Cette définition introduit assez naturellement les grandes familles de modèles que l'on va pouvoir rencontrer. Nous allons ici tenter de dresser un portrait des modèles. L'objectif n'est pas d'être exhaustif mais plutôt de donner une représentation générale des différentes branches explorées au cours de ce

travail. Cette hiérarchie est inspirée de [CDK05]. Pour plus d'exhaustivité, [DDS87] présente une énumération des modèles selon 3 facteurs d'asynchronie.

2.1.1 Modèles des interactions

Nous pouvons classer ici tous les modèles qui définissent la manière dont nos entités interagissent. Il est possible de définir deux sous catégories : les modèles décrivant les acteurs des interactions et les modèles décrivant les interactions elles-mêmes. Nous prendrons l'exemple de l'impact du temps dans chaque catégorie.

Acteurs des interactions

Ici entrent toutes les modélisations des réseaux utilisés par les systèmes répartis. De façon générale, on modélise un système réparti par un graphe $G(V, E)$ où chaque *noeud* $i \in V$ représente une entité (dans la suite, noeud et entité désigneront le même objet), et chaque arc $(i, j) \in E$ représente une interaction de i et j. On dit alors que j est *voisin* de i. Cette catégorie « acteurs des interactions » va décrire les modèles courants décrivant quel i rencontre quel j.

Relation au temps : Dynamisme. L'évolution du graphe au cours du temps est un paramètre important. Ainsi, on distinguera les modèles selon leurs rapports au temps. Une majeure partie des modèles suppose un réseau et un graphe constant au cours du temps. On parle alors de système *statique*. A l'opposé, certains travaux (notamment les travaux orientés « système ») prennent en compte l'évolution du graphe au cours du temps. On parle alors de système *dynamique*. Ce dynamisme peut avoir plusieurs sources. Citons :

- la mobilité des entités. Dans cette famille de modèles on manipule des entités mobiles (dans un plan par exemple) communiquant sans fil. Ainsi le voisinage d'un noeud est constitué de l'ensemble des noeuds à portée radio. Ce voisinage évolue puisque les noeuds sont mobiles. Au niveau graphe, on peut le traduire par un ensemble E fonction du temps.
- les aléas des couches de communication. Certains noeuds ne peuvent plus communiquer entre eux à partir d'un certain moment. Cela modélise par exemple la panne d'un routeur connectant une partie des entités aux autres. Au niveau graphe, on retire des arêtes.
- le *churn*, ou va-et-vient des noeuds. Il s'agit ici de modèles fréquemment utilisés pour décrire le comportement des entités composant les réseaux pair à pair publics, où en permanence de nouvelles entités arrivent et d'anciennes entités partent, à la manière des voyageurs dans une gare. Au niveau du graphe, on ajoute et on retire des sommets (et donc des arêtes).

Le problème des modèles avec prise en compte du dynamisme est qu'ils sont très nombreux et que le dynamisme peut prendre des formes très variées. Le churn [RGRK04] est par exemple source de nombreuses recherches et a des impacts très variables selon de nombreux paramètres : taux d'arrivée, taux de départ, distribution de la probabilité de départ selon l'âge des entités, etc.

Déroulement des interactions

Nous allons ici présenter les modèles typiques décrivant le déroulement des interactions. Généralement, ces modèles capturent le comportement du médium de communication.

Symétrie des interactions. La majorité des modèles modélise une interaction de façon symétrique. C'est effectivement souvent le cas dans les réseaux étudiés. Ainsi si a peut communiquer avec b il y a de grandes chances pour que le contraire soit possible. Cela se traduit par un graphe G symétrique, ou encore par l'équivalence sur l'ensemble des arcs : $(i, j) \in E \Leftrightarrow (j, i) \in E$.

Relation au temps : Synchronie et asynchronie. Un des plus grands discriminants des modèles répartis est le rapport entre les interactions et le temps. C'est ici que se trouve la différence entre les systèmes *synchrones* et les systèmes *asynchrones* : un système est dit synchrone s'il existe une borne haute finie sur le temps de transit des messages (la latence). Autrement dit, s'il existe $\tau > 0$ tel que $\forall p_i, p_j \in \Pi$, tout message émis par p_i à l'instant δ pour p_j sera reçu avant l'instant $\delta + \tau$. S'il n'existe pas de borne, alors le système est dit asynchrone.

Le résultat le plus célébré par la communauté des systèmes répartis [FLP85] montre que dans un système asynchrone comportant un processus pouvant crasher, il est impossible de faire s'accorder tous les processus corrects sur une valeur (le problème du consensus). En effet, la combinaison des crashs et de l'asynchronie a une conséquence dramatique : un processus attendant un message ne sait jamais s'il s'agit d'un message lent ou d'un message qui n'a jamais été émis puisque le processus émetteur a crashé.

L'hypothèse de l'asynchronie est une hypothèse contraignante pour le concepteur de système réparti. Néanmoins, réaliser des protocoles pour systèmes asynchrones a deux intérêts pratiques :

- La robustesse. Un protocole conçu pour un système asynchrone fonctionnera bien en système synchrone. Le contraire est faux. Ainsi pour utiliser un protocole synchrone, il faut faire l'hypothèse d'un temps de communication τ maximum. Si ce temps choisi est trop faible, c'est le système qui est en péril.
- La performance. Le paramètre τ va rythmer le système. On appelle souvent *ronde* un échange global d'information par tous les processus (c'est-à-dire la succession : « envoyer un message à tous », « recevoir tous les messages »). Afin de s'assurer que tous les processus possèdent les mêmes informations, la durée de cette ronde est souvent fixée à τ. Ainsi un τ trop grand augmente la durée des rondes et ralentit le système. Cet argument est développé dans le prologue de [CHC04] : la lumière mettant $70ms$ a parcourir la moitié du globe, un système synchrone réparti à l'échelle de la planète n'ira jamais plus vite.
  De plus, beaucoup de réseaux ont une distribution de temps de latence qui suit une loi normale [ZGG05]. Autrement dit, dans ces réseaux, la majorité des messages arrivent vite, mais parfois les messages mettent beaucoup plus de temps à arriver. Pour pouvoir supposer ces systèmes synchrones, il faut alors attendre tous les retardataires et fixer un τ beaucoup plus élevé que la latence moyenne.

Une solution développée pour sortir de cette impasse consiste en la réalisation de protocoles *indulgents* [Gue00]. Ces algorithmes sont conçus de façon à réaliser une tâche dans un modèle donné (ici un modèle synchrone) en résistant aux périodes où le système ne se comporte pas conformément au modèle (périodes d'asynchronie).

2.1.2 Modèles des entités

Comme les communications, il existe différents modèles décrivant les entités. Il existe pléthore de modèles décrivant les capacités et les limitations des entités, mais aussi les capacités et les limitations des possesseurs de telles entités : c'est ainsi que cette famille peut contenir les modèles comportementaux de téléchargement et de partage des utilisateurs de téléchargement pair à pair. Ces modèles sortent du cadre de notre étude et ne seront pas abordés. A titre d'exemple, nous présentons ici des modèles plus simples, qui capturent les formes de défaillance des entités. Enfin nous évoquerons les modèles capturant la connaissance des entités.

Entités faillibles

Afin de parvenir à la robustesse annoncée en introduction, il est important de concevoir des systèmes dont le bon fonctionnement n'est pas mis en danger par la défaillance d'une ou plusieurs

entités le composant. Pour pouvoir concevoir des systèmes robustes aux défaillances, il faut alors modéliser ces défaillances.

Le modèle le plus courant de défaillance est le modèle *crash-stop* ([SS83]). Dans ce modèle, une entité faillible va, à un moment quelconque, cesser définitivement toute activité. C'est-à-dire qu'elle va cesser d'émettre et de recevoir des messages. Les entités non faillibles sont dites *correctes*. A un moment donné, une entité qui n'a pas crashée est dite *vivante*.

En général, on suppose qu'un système contient un nombre maximal d'entités faillibles, souvent noté t. Les protocoles dont le fonctionnement est prouvé dans les systèmes ayant ce modèle sont dits *t-resilients*. Dans un système asynchrone, lorsque $t = n - 1$, c'est-à-dire lors qu'au pire le système ne contient plus qu'une seule entité, on parle de protocole *wait-free* : l'entité ne peut en effet attendre aucune réponse puisqu'elle peut instantanément se retrouver seule entité vivante du système.

Byzantins

Le modèle crash-stop vu précédemment n'est pas l'unique modèle décrivant la défaillance d'une entité. Il existe de nombreux types de défaillance, chacune ayant un impact différent sur la décidabilité du système (voir [Cri91]). L'autre grand modèle est le modèle dit des byzantins [LSP82]. Dans ce modèle, les entités faillibles que l'on appelle alors *byzantins*, peuvent se comporter arbitrairement.

Traditionnellement, ce modèle de crash est associé à une notion d'adversité : un ennemi cherche à empêcher le bon fonctionnement du système, et ce par l'intermédiaire d'une ou plusieurs entités qu'il contrôle. Notons d'ailleurs que cette définition permet tout à fait la concorde (c'est-à-dire la collaboration de plusieurs byzantins). Cet adversaire est supposé très puissant : il connaît tout, y compris les messages transitant entre deux entités correctes par exemple, il a une capacité de calcul très grande, et beaucoup de chance.

Cependant, il est intéressant d'observer que ce modèle est au final l'absence de toute hypothèse sur le mode de défaillance. En effet, tout ce que dit la spécification du byzantin est que le comportement est arbitraire. Cela veut dire « faillir de n'importe quelle manière ». Il est alors commode d'imaginer le pire cas, c'est-à-dire un adversaire puissant. Mais l'utilité du modèle du byzantin dépasse ce contexte d'adversité : ce modèle capture par exemple les erreurs de programmation ou encore la défaillance des circuits électroniques d'une entité.

Connaissance des processus

Dans un système réparti, la connaissance des entités va conditionner grandement la complexité de certaines tâches. Il est par exemple courant de supposer que tous les processus sont différenciables [Ang80] ou ont une identité unique, et ne pas faire cette hypothèse a de lourdes conséquences (voir *e.g.* [BR09]).

De même, alors que la recherche fondamentale sur les problèmes d'accord suppose généralement que les entités participant à l'accord connaissent n (le nombre d'entités participant à l'accord), cette hypothèse est forte dès lors que l'on cherche à modéliser les réseaux pair à pair large échelle. D'autant qu'avec le dynamisme, la connaissance de ce nombre exact périme vite.

D'une façon générale, il est important de capturer de façon formelle la connaissance à laquelle les entités du système ont accès. Il est aussi important de capturer les variations de la véracité de cette information. Ce sera le rôle des *oracles* que nous aborderons plus tard.

Messages

Notons enfin que [DDS87] introduit une troisième source d'asynchronie : celle des messages. En effet certains modèles ne font pas l'hypothèse de communications asynchrones mais de canaux dits *fifo* (*F*irst *I*n *F*irst *O*ut) : ce sont des canaux où les messages sont livrés dans l'ordre où ils ont été envoyés.

2.2 Couches et boites noires

Nous avons vu que les modèles permettent d'établir ce qui est fourni aux entités. Comme décrit précédemment, l'intérêt de cette approche est la généricité : les exigences sont portées sur les résultats, pas sur les méthodes, ce qui laisse, à la manière de la recette de cuisine, la liberté de l'implémentation au concepteur du système.

Dans ce sens, on peut qualifier cette vision de *boite noire* : chaque solution est spécifiée en termes de besoins (c'est-à-dire quel hypothèses sont nécessaires à cette boite) et de services (qu'est-ce que cette boite fournit si ses besoins sont remplis). Cette approche est courante dans le domaine informatique.

Une des boites noires que nous rencontrerons souvent dans cette étude sont les *oracles détecteurs de défaillances* [CT96]. Ces oracles sont utilisés pour enrichir les modèles afin de pouvoir y résoudre plus de problèmes (*i.e.* augmenter leur calculabilité). Un détecteur de défaillances fournit des indications (potentiellement fausses) aux processus du système.

L'objectif final est de fournir au concepteur de systèmes répartis une trousse à outils composée de boites qu'il va pouvoir assembler afin d'obtenir pour son système les fonctionnalités qu'il désire.

2.2.1 Hiérarchie de modèles

Les modèles ne sont pas tous égaux. Si l'on voit le modèle comme une série d'hypothèses, il est alors aisé d'imaginer la relation d'ordre partiel qui les lie. Imaginons deux modèles M_1 et M_2 capturant respectivement des ensembles de systèmes S_1 et S_2. On dit de M_1 qu'il est plus *fort* que M_2 ssi $S_2 \subset S_1$. Par exemple, le modèle synchrone est plus fort que le modèle asynchrone. Il est important de remarquer que c'est une relation partielle. On retrouve cette définition dans les détecteurs de défaillances [CT96]. Ainsi le modèle de crashs byzantins n'a rien à voir avec le modèle asynchrone.

Un des grands objectifs de la recherche en systèmes répartis est de trouver des modèles utiles et de définir les modèles les plus faibles permettant de résoudre un problème donné. En effet plus un modèle est fort, plus les systèmes décrits par ce modèle sont rares. D'un point de vue pratique utiliser un modèle fort n'est pas sans conséquences : plus un modèle est fort, plus il introduit d'hypothèses, et donc plus l'une de ces hypothèses a des chances d'être violée. La violation des hypothèses n'est pas souhaitable, puisque la majorité des algorithmes ont un comportement non spécifié en dehors du modèle dans lequel ils sont censés fonctionner.

2.3 Enjeux de la conception en couches

Concevoir des systèmes répartis revient à assembler des boites. La recherche a pour objectif de fournir ces boites, telles des briques, dans l'optique de leur assemblage. Faisons un parallèle avec la réalisation d'une maison : elle est composée d'un toit, supporté par des murs, qui eux-mêmes reposent sur des fondations, ancrées dans le sol. Les études sur les murs ont plusieurs objectifs :

Fiabilité les murs doivent demeurer solides malgré les intempéries

Economie la construction et l'entretien du mur doivent être les plus faibles possible.

Généricité les murs doivent s'adapter sur le plus grand éventail de fondations possible.

Adaptabilité les murs doivent permettre de soutenir une grande variété de toits.

Remarquons qu'il n'existe pas d'idéal absolu dans cette description. Ainsi, fiabilité et économie sont des qualités souvent antagonistes et l'optimum dépend du contexte : un mur suffisamment solide pour un bord de mer sera perçu comme un gâchis de matériaux dans une prairie abritée.

Dans le cadre des systèmes répartis, ce sont les mêmes critères qui vont orienter les recherches. Nous allons ici détailler les étapes de l'évaluation des protocoles répartis.

2.3.1 Complexité

Tout comme dans la construction l'objectif est de réaliser des protocoles économes. Le coût d'un protocole réparti est multiple :

Messages l'échange de messages entre entités est souvent une ressource critique. Le but est ici de minimiser la sollicitation de médium de communication.

Mémoire et puissance de calcul les entités ont des capacités limitées. Ceci est particulièrement vrai dans le cadre des réseaux de capteurs.

Temps le temps est une ressource limitée, d'autant que les réseaux réels ne sont pas stables au cours de celui-ci.

En général, on exprime ces coûts en fonction des paramètres d'entrée du problème. Ainsi en général, la complexité d'un problème est liée au nombre n d'entités du système dans lequel on cherche à le réaliser. D'un point de vue théorique, on ne présente pas le coût du problème mais plutôt sa variation en fonction de l'augmentation de la taille du système.

Des techniques existent pour réduire ces coûts. Plus les garanties quant à la qualité d'un mur sont fortes plus le coût de ce mur sera élevé. Il en va de même dans les systèmes répartis. On appelle *déterministe* un protocole dont le résultat est exact et entièrement déterminé par les données d'entrée. C'est un résultat assuré par une garantie forte. A l'opposée, il existe des protocoles *non déterministes*, reposant en partie sur le hasard. Ces processus produisent des résultats exacts avec une probabilité 1 (pourvu que l'on attende suffisamment longtemps), comme par exemple [BO83], soit des estimations du résultat [GKMM07]. Sacrifier le déterminisme est une méthode qui peut s'avérer rentable si l'exactitude ou la promptitude d'obtention du résultat ne sont pas des facteurs critiques, ou s'il n'existe pas d'autre solution [BO83].

Prenons un exemple : on souhaite dénombrer le nombre d'entités composant un réseau statique synchrone. Plusieurs solutions :

- Chaque entité émet un message à l'intégralité des autres entités. Le nombre d'entités est le nombre de messages collectés. S'il y a n entités le coût est de $n(n-1)$ messages. Le résultat est exact et obtenu en une ronde d'échanges.
- Une entité chargée du comptage est désignée. Toutes les entités lui envoient un message, l'entité chargée du comptage relaie le résultat à tous. Le coût est de n messages pour la collecte, plus le coût de la désignation d'une entité centrale, qui peut être élevé si l'on désigne plusieurs fois par malchance une entité qui tombe en panne. Le temps d'obtention du résultat exact dépend lui aussi du bon choix de l'entité centrale.
- Le nombre d'entités du système est évalué à l'aide de probabilités [GKMM07] (méthode dite de capture-recapture). Un résultat approximatif est obtenu rapidement.

2.3.2 Fiabilité

Il est important d'étudier la fiabilité des solutions proposées. Pour cela nous utilisons deux méthodes :

- Lorsque le modèle est suffisamment simple pour être manipulé mathématiquement, la fiabilité est assurée par des preuves.
- Lorsque le modèle est trop complexe pour permettre une preuve, l'algorithme et l'environnement (c'est-à-dire le modèle) sont simulés. Cela permet notamment de découvrir l'impact de certains paramètres du modèle sur le comportement de l'algorithme.

2.3.3 Structure

Suivons la métaphore du mur à la lumière des deux propriétés « généricité » et « adaptabilité ». Ce que l'on cherche à exprimer ici est la capacité du mur à être adapté à ce sur quoi il repose et à ce qu'il supporte. Quelles sont les propriétés importantes pour cette adaptation ? La couleur du mur, ou du toit, importe peu. Pas plus que le matériau dans lequel la fondation est réalisée. Par contre, la planarité des fondations est une propriété que l'on imagine importante, tout comme la forme du toit, ou encore leurs rigidités respectives. En d'autres termes ce ne sont pas les propriétés intrinsèques qui sont ici importantes, mais plutôt les propriétés structurelles des éléments interagissant avec notre mur.

La structure est un mot aux nombreuses significations. La définition de *structure* que nous allons manipuler ici est celle que Nicolas Bourbaki donne dans « L'Architecture des mathématiques » :

> Le trait commun des diverses notions désignées sous ce nom générique, est qu'elles s'appliquent à des ensembles d'éléments dont la nature n'est pas spécifiée ; pour définir une structure, on se donne une ou plusieurs relations, où interviennent ces éléments [...] ; on postule ensuite que la ou les relations données satisfont à certaines conditions (qu'on énumère) et qui sont les axiomes de la structure envisagée. Faire la théorie axiomatique d'une structure donnée, c'est déduire les conséquences logiques des axiomes de la structure, en s'interdisant toute autre hypothèse sur les éléments considérés (en particulier, toute hypothèse sur leur « nature » propre).

Les modèles décrits précédemment comportent de telles structures sous plusieurs formes. Ainsi le modèle de graphe de communication décrit une structure évidente. Moins évidente, l'hypothèse « tous les noeuds ont une identité unique » est une structure dont nous pouvons expliciter l'axiome : si id_i désigne l'identité du processus i, nous avons $\forall i, j \in \Pi, id_i \neq id_j$. Il est ainsi possible de regarder tous les modèles de communication ou de connaissances des entités comme des structures.

Il en va de même avec les services rendus : élire un leader [AR07], créer un arbre couvrant [BPBRT09] ou doter les noeuds d'un système de routage [KMR$^+$09] revient à construire des structures que l'on propose aux applications des couches supérieures.

L'objectif de ce travail va être d'observer les systèmes répartis à la lumière des structures. Le constat de départ est simple : les modèles de départ sont des structures, les services rendus sont des structures, et ces services sont rendus à l'aide d'hypothèses effectuées sur les structures sous-jacentes : un algorithme réparti transforme donc les structures.

L'étude des systèmes répartis va se faire selon trois axes :

- Le premier chapitre se concentrera sur l'étude de l'impact des structures sous-jacentes à un algorithme. Nous verrons que ces structures impactent fortement des protocoles même très simples. A partir de ce constat, nous chercherons à caractériser de façon répartie ces structures, afin de donner aux entités conscience de la structure qu'elles composent.

- Le second illustrera comment les structures du modèle sont transformées et exploitées afin de produire une nouvelle structure pratique (le service). Cela nous permettra d'observer la transformation d'une structure implicite (formée par l'instance des moyens de communication) en une autre structure implicite (chaque entité connaît son état vis-à-vis de la structure mais n'a pas de vision globale de la structure).
- La troisième partie se concentrera sur la concentration et l'exploitation d'une structure particulière : le système de coordonnées. Cette structure, par opposition aux structures évoquées précédemment, est explicite. Nous verrons comment exploiter ce caractère.

Chapitre 3

Étude des structures naturelles des systèmes répartis

Nous avons vu en introduction que les systèmes répartis sont constitués d'entités communiquantes. Nous avons vu qu'une exploitation du système réparti, c'est-à-dire un moyen de concevoir des applications réparties, consiste à bâtir et à maintenir des structures dans ces systèmes. Pour cela, nous allons utiliser ce qui existe déjà dans le système réparti : la capacité des entités à communiquer. Ainsi, les structures que nous allons bâtir reposent sur la capacité des entités à communiquer.

Nous allons dans ce chapitre nous concentrer sur cette capacité à communiquer. Pour reprendre l'image de la construction, celle-ci va constituer la base, ou la fondation du mur "application" que nous souhaitons réaliser. Il est en ce sens important de bien comprendre les influences qu'auront ces fondations sur la qualité du mur que nous souhaitons y bâtir.

Toutes les entités du système réparti ont la capacité de communiquer. Cependant, les entités sont bien souvent inégales devant cette capacité : certaines communiquent facilement, d'autres communiquent plus difficilement, par exemple parce que leur accès au réseau est limité. Nous allons illustrer cette différence de capacité de communication à l'aide de deux études que nous présenterons dans ce chapitre

- Dans le premier système, chaque entité peut envoyer des messages à la totalité du système (c'est-à-dire qu'elles ont accès à une primitive de diffusion, nommée *broadcast*). Cependant, les messages ainsi échangés n'arrivent pas aux mêmes instants à toutes les entités. Ainsi les messages transitent vite entre certaines entités et lentement entre d'autres. C'est un phénomène courant sur la quasi-totalité des réseaux, ne serait-ce que parce que le temps de transit des messages est souvent corrélé à la distance physique parcourue par ceux-ci, ainsi des entités physiquement proches vont souvent communiquer plus rapidement que des entités éloignées.
- Dans le second système, les entités n'ont pas accès à une primitive de diffusion. Chaque entité connaît un petit nombre d'autres entités, que nous appellerons ses *voisins*, et c'est le réseau formé par les connaissances des entités qui est l'objet de l'étude. Dans ce système aussi, les entités sont inégales devant l'action de communication, mais ici le facteur discriminant n'est plus la vitesse de communication mais la capacité à trouver de bons interlocuteurs.

Le fait que les entités soient inégales face à la communication crée dans le système une structure (formée par l'ensemble de relations d'interaction). Le concepteur d'applications réparties n'a souvent pas le contrôle sur cette structure (il ne peut pas accélérer les paquets, rajouter des lignes haut débit, ou déplacer des capteurs), elle lui est imposée. Il est alors d'autant plus important de comprendre et maîtriser l'influence que ces structures exercent sur les applications réparties.

Dans ce chapitre, nous nous intéressons à deux aspects de l'influence des structures sous-jacentes :

- d'un côté, il s'agit de comprendre leur impact sur une application répartie simple. Ainsi dans la première partie, nous cherchons à fournir à chaque entité le nombre des entités vivantes qui composent le système. Nous verrons comment cette application simple est fortement impactée par la structure du réseau sous-jacent.
- de l'autre, il s'agit d'exploiter cet impact pour permettre aux entités composant le système d'estimer la qualité du réseau de communication sous-jacent. L'objectif de la deuxième partie est de fournir aux entités un indicateur de la qualité de la topologie qu'elles constituent.

3.1 Estimation du nombre de processus vivants dans un système asynchrone

3.1.1 Motivation

Incertitude des systèmes asynchrones

Dans cette partie, nous supposons un système asynchrone à transfert de messages. A l'opposée, un système synchrone est un système dans lequel les transferts de messages sont synchrones, c'est-à-dire qu'il offre une borne haute sur le temps de transfert des messages. Cette borne est de plus connue des entités composant le système, ce qui leur permet de l'utiliser dans leurs calculs. Supposons maintenant que ces entités peuvent crasher (par des crashs de type crash-stop). Une question centrale est d'obtenir une estimation du nombre de processus vivants dans le système. Ici, dans le cas d'un système synchrone, un protocole trivial permet d'obtenir une estimation de ce nombre : supposons que le temps de traitement d'un message est nul (on peut toujours l'inclure dans le temps de transfert). Alors périodiquement un processus diffuse un message de type *ping* auquel les autres processus répondent par retour de message, puis déclenche un temporisateur. Puisqu'il connaît la durée maximale d'un aller-retour de message dans le système, il peut conclure qu'après cette durée tous les processus n'ayant pas répondu au message sont crashés. Notons que ce n'est qu'une sous-estimation du nombre de processus crashés, puisque certains processus ont pu crasher depuis l'émission de leur réponse.

A l'opposée, dans un système dit *asynchrone*, aucune hypothèse n'est faite sur le temps maximal de transfert d'un message (si ce n'est que celui-ci est fini)[AW04, Lyn96]. On ne peut ainsi pas parler de temps maximal de transfert. Ainsi, quelle soit la durée qu'attend un processus, il ne peut rien conclure quant au nombre de processus crashés. L'impossibilité de faire la différence entre un processus dont le médium de communication est lent et un processus crashé est une des difficultés majeures de la conception de protocoles répartis dans des systèmes asynchrones soumis aux crashs. Cette impossibilité est à la source de l'impossibilité de résoudre le problème du consensus dans ces systèmes, un des résultats les plus connus du domaine [FLP85].

Prenons un système réparti asynchrone classique. Soit n le nombre d'entités, ou processus, le composant (nous considérons un système statique). Une manière classique de contourner l'impossibilité soulignée dans le paragraphe précédent consiste à enrichir le modèle d'une hypothèse supplémentaire : on suppose qu'il existe un paramètre t ($t < n$) qui est la borne haute du nombre de processus pouvant crasher. Ce paramètre t peut être vu comme un pari sur l'évolution du système.

Dans ce modèle, considérons un processus p cherchant à obtenir de l'information du maximum possible de processus. A cette fin, p envoie une requête et le paramètre t du modèle est utilisé pour définir une limite logique à l'attente de p : il attend jusqu'à recevoir $n - t$ réponses, sans recourir à l'utilisation de temporisateurs.

Si l'on définit la « qualité de réponse » à une requête donnée comme le nombre de réponses

reçues pour cette requête. Le modèle classique possède, à la lumière de cette qualité de réponse, deux faiblesses. Soit f le nombre effectif de crashs dans le système :

- si $f < t$, puisque p n'attend que $n-t$ réponses, il rate $t-f$ réponses. Cela est potentiellement très pénalisant si f est petit. La qualité de réponse est toujours de $n-t$ alors qu'elle pourrait être de $n-f$.
- si $f > t$, p se bloque : il attend les $f-t$ réponses manquantes, qui n'arriveront jamais.

Dans le premier cas, la qualité de réponse est réduite. Dans le second cas, t n'était pas la borne haute du nombre de crashs dans le système. Bien sur, il est possible pour pallier au problème de choisir un t plus grand, mais c'est au détriment de la qualité de réponse comme souligné dans le premier cas. Ainsi, trouver la bonne estimation de t est un défi lorsque l'on souhaite à la fois de bonnes qualités de réponse tout en évitant le bloquage du système.

3.1.2 Travaux connexes

Le problème des crashs de processus est un problème qui a concentré beaucoup d'efforts de la communauté. De l'impossibilité du consensus [FLP85], diverses méthodes ont été développées pour contourner le problème, par l'utilisation de détecteur de défaillances [CT96] (voir [Ray05] pour une étude de cette approche), par l'ajout de synchronie [DLS88], ou par l'utilisation de protocoles aléatoires [BO83, Rab83].

Pour les systèmes large échelle, [RMH98] propose un service de détection de crashs basé sur le gossip. [Fet03] présente la détection de défaillances dans les systèmes dits temporisés (définis dans [CF99]). [KSC03] étudie la dégradation de performance de services face à des variations erratiques de charge et des crashs. [HDYK04] présente une approche nouvelle reposant sur des niveaux de suspicions adaptés dynamiquement en fonction de l'état du réseau.

3.1.3 Modèle

Nous allons ici définir plus formellement le modèle utilisé pour ces travaux. Il s'agit d'un système asynchrone composé d'un ensemble $\Pi = \{p_1, \ldots, p_n\}$ de n processus (ou entités) communiquant par échange de messages. Chaque processus a sa propre vitesse et les messages échangés le sont en un temps fini mais **non borné**.

Le réseau est supposé fiable, c'est-à-dire que chaque message émis sera reçu une fois exactement : il n'y a ni corruption, ni perte, ni duplication. Les processus sont moins fiables que le réseau : ils peuvent crasher, c'est-à-dire qu'il stoppe prématurément tout calcul et toute communication. Lors d'une exécution donnée, un processus qui crashe est dit défaillant, un processus qui ne crashe pas est dit *correct*. Un processus est dit *vivant* s'il n'a pas encore crashé.

Les processus ont à leur disposition une opération BROADCAST(m), avec m un message. C'est une opération qui n'est pas atomique : c'est un raccourci pour « **for each** $p_j \in \Pi$ **do** send(m) to p_j **end do**. ». Cela veut dire qu'un processus peut crasher au cours de l'exécution de cette opération, le message m sera alors reçu par un sous-ensemble arbitraire de Π.

Chaque processus possède une horloge locale. Il obtient la date actuelle en invoquant la méthode local_clock(). Ces horloges ne sont pas synchronisées : elles peuvent avoir des valeurs différentes au même instant d'invocation.

Chaque horloge locale est supposée sans dérive (c'est-à-dire qu'elle ne se dérègle pas). Il est possible de considérer des horloges dont la dérive serait bornée, mais cela introduirait des lourdeurs dans la présentation dans la solution. Nous n'utilisons les horloges locales que localement, afin d'estimer la durée écoulée entre deux actions locales. Nous supposons les horloges suffisamment précises pour s'incrémenter entre deux étapes locales consécutives.

Comme nous l'avons vu plus haut, il est nécessaire d'introduire des hypothèses supplémentaires sur le modèle si l'on souhaite réaliser des protocoles utiles dans un système asynchrone. De telles

hypothèses, lorsqu'elles sont satisfaites, permettent la réalisation de protocoles corrects et non bloquants. Nous avons déjà rencontré une telle hypothèse : t.

Malheureusement, t est statique, c'est-à-dire qu'il est définit une fois pour toutes. A la place, nous proposons un modèle où les processus ont à leur disposition une fonction $\alpha()$. Cette fonction prend une durée Δ en paramètre et retourne au processus appelant un entier compris entre 0 et $n-1$, qui est une estimation du nombre de processus pouvant crasher pendant Δ unités de temps. Il est possible de voir $\alpha()$ comme une anticipation du nombre de crashs du système. Cette anticipation est à caractère beaucoup plus dynamique que le simple paramètre t.

Afin d'être utile $\alpha()$ doit remplir les conditions suivantes :

- $\Delta_1 \geq \Delta_2 \Rightarrow \alpha(\Delta_1) \geq \alpha(\Delta_2)$ (non décroissante).
- Elle doit inéluctablement croître : plus le temps passe plus les processus ont des chances de crasher. Plus formellement : $\forall \Delta_1$ tel que $\alpha(\Delta_1) < n-1$, $\exists \Delta_2 > \Delta_1$ tel que $\alpha(\Delta_2) > \alpha(\Delta_1)$.

D'un point de vue pratique, $\alpha()$ peut être définie de multiples manières : par exemple par l'observation des précédentes exécutions du système, ou encore théoriquement à l'aide d'une analyse des probabilités de crashs. C'est dans les systèmes où les pannes ont très peu de chances d'être corrélées que cette approche est la plus efficace : imaginons un système où tous les processus peuvent crasher. Dans le modèle classique, on a $t = n-1$, ce qui condamne les processus à ne jamais attendre (c'est un protocole *wait-free*) et complique considérablement l'implémentation d'un quelconque protocole. Dans le nouveau modèle, la fonction $\alpha()$ dépend idéalement de la corrélation entre ces crashs : si tous les processus crashent indépendamment, il est peu probable qu'ils crashent tous au même instant, et $\alpha()$ prendra de petites valeurs pour de petites durées, ce qui permet de réaliser des échanges d'informations jusqu'à ce que le nombre de crashs soit *effectivement* trop important.

Notons qu'il est possible, dans le modèle avec $\alpha()$, de retrouver le modèle classique avec t : il suffit de définir $\forall \Delta \in \mathbb{R}^+, \alpha(\Delta) = t$. Cependant, nous verrons que le fait que $\alpha()$ ne tende pas vers $n-1$ n'élimine pas la possibilité d'attentes bloquantes : on retrouve alors le défaut du modèle classique.

Nous avons ici volontairement gardé la définition de $\alpha()$ simple afin de reste pédagogiques, mais il est possible d'imaginer beaucoup plus complexe et beaucoup plus pratique. Par exemple, pour les systèmes vastes, des $\alpha_i()$ locaux que chaque processus pourrait alimenter de sa propre expérience du système. De même, la fonction $\alpha_i()$ n'a pas besoin de rester identique tout au long de l'exécution du système. On peut ainsi imaginer un système supposant initialement un taux de crashs de 6 processus par unité de temps. Après un certain temps si l'administrateur observe que le système est fiable et se comporte bien, il peut descendre le taux à 4 processus par unité de temps.

3.1.4 Estimation du nombre de processus vivants

Dans cette partie nous présentons un protocole qui procure à chaque processus un ensemble de processus estimés vivants. Puisque les processus connaissent Π, l'ensemble des processus, il est possible d'en déduire l'ensemble des processus estimés crashés. Nous allons présenter deux protocoles afin de faciliter la présentation : le premier repose sur une horloge globale, le second utilise des horloges locales non synchronisées pour remplacer cette horloge globale. Nous supposons que le calcul local des processus est instantané, le temps nécessaire à celui-ci pouvant être intégré au temps de transit des messages (qui lui prend un temps arbitraire).

Protocole basé sur les horloges globales

Ici nous supposons donc que le système fournit aux processus une horloge globale lue par les processus en invoquant la fonction global_clock(). Chaque processus exécute régulièrement

```
operation estimate():
1:      start_time_i ← global_clock();
2:      broadcast QUERY();
3:      wait until (|est_i.set| − β)) corresponding RESPONSE(rec_from_j) have been received
4:          where β = α(global_clock() − est_i.date) is continuously evaluated;
            % If any, when they arrive, the other corresponding response messages are discarded %
5:      rec_from_i.date ← start_time_i;
6:      rec_from_i.set  ← the set processes from which p_i has received RESPONSE() at line 03;
7:      est_i.date ← min over the rec_from_j.date received at line 03;
8:      est_i.set  ← ⋃ of the rec_from_j.set received at line 03;
9:      return(est_i.set)

background task T:
 when QUERY() is received from p_j do send RESPONSE(rec_from_i) to p_j end_do
```

FIG. 3.1 – L'opération estimate() (version basée sur des horloges globales)

la fonction estimate() dont le code est donné en figure 3.1. C'est cette opération qui renvoie l'ensemble des processus estimés vivants (ligne 09).

Chaque processus p_i utilise trois variables locales :

- La variable est_i, composée de deux champs. Le champ $est_i.set$ contient l'ensemble des processus que p_i estime vivants à la date courante. Tous les processus contenus dans cet ensemble étaient vivants à la date $est_i.date$.
- La variable rec_from_i, elle aussi composée de deux champs : le champ $rec\_form_i.set$ est le dernier ensemble de processus de qui p_i a reçu des réponses. De la même manière, $rec\_form_i.date$ est la date à laquelle tous les processus de $rec\_form_i.set$ étaient vivants.
- $start\_time_i$ est une variable qui contient la dernière date à laquelle p_i a envoyé un message. C'est elle qui va permettre d'établir $rec\_form_i.date$.

Nous allons maintenant décrire le comportement des processus. Régulièrement, chaque processus p_i exécute la fonction estimate(). Cette opération est un mécanisme de requête/réponse, comme celui introduit dans [MMR03] : p_i envoie une requête (ligne 02) et attend les réponses correspondantes (ligne 03). Notons que chaque requête possède un identifiant, qui est repris dans les réponses correspondantes afin de pouvoir associer les requêtes et les réponses. Nous avons volontairement omis ces identifiants afin de ne pas surcharger le code du protocole. Notons qu'une réponse arrivant après la fin de l'attente lui correspondant est ignorée. Ensuite, p_i exploite les réponses afin de mettre à jour ses variables rec_from_i et est_i de la manière suivante :

- $rec\_from_i.set$ est l'ensemble des processus de qui p_i a reçu une réponse pendant sa période d'attente (lignes 03-04). Notons qu'indépendamment du temps de transit du message, les processus ayant répondu à la requête de p_i étaient vivants à l'instant ou p_i émettait la requête (à savoir $start\_time_i$). $rec\_from_i.date$ prend donc la valeur de $start\_time_i$ (lines 05).
- Lorsqu'un processus p_j répond à la requête de p_i, sa réponse contient la valeur courante de rec_from_j (voir la partie « background task »).
  L'union des ensembles $rec\_from_j.set$ reçus par p_i est utilisée pour établir la nouvelle valeur de $est_i.set$ (ligne 08), c'est-à-dire l'ensemble de processus que p_i croit vivants. Formulé autrement, p_i croit vivant l'ensemble des processus dont il a reçu des informations à l'étape précédente (puisqu'il répond toujours à sa propre requête) et l'ensemble des processus dont ces processus ont reçu des informations.
  Comme expliqué précédemment, pour tout j, l'ensemble des processus de $rec\_from_j.set$ étaient vivants à l'instant $rec\_from_j.date$, donc tous les processus de l'ensemble $est_i.set$ étaient vivants à l'instant $\min(rec\_from_{j_1}.date, \ldots, rec\_from_j.date, \ldots, rec\_from_{j_x}.date)$

(où chaque $rec\_from_{j_k}.date$ correspond à une réponse reçue par p_i à la requête émise précédemment) ; $est_i.date$ est donc positionné à cette date (ligne 07).
Ici encore, nous avons utilisé l'union et le minimum comme le moyen de réunir les *estimats* reçus. Soit K_i l'ensemble des processus répondant à p_i à une ronde donnée. Il est possible de faire mieux en réunissant ces ensembles de manière plus précise : imaginons que tous les processus $k \in K_i$ répondants véhiculent la même liste de processus $rec\_from.set$, il serait alors logique de considérer comme date de référence pour $est_i.date$ la date la plus avancée (*i.e.* $\max_{k \in K}(rec\_from_k.date)$), et non la date minimum. Autrement dit, considérer la date minimum de l'ensemble des $rec\_from_k.date$ est une façon sûre, mais non optimale, de combiner les estimats.

Maintenant, il reste à spécifier combien de réponses p_i doit attendre (lignes 03-04). C'est ici que la fonction $\alpha()$ s'avère utile. Tant qu'il attend, p_i évalue $\beta = \alpha(\mathsf{global\_clock}() - est_i.date)$ (ligne 04). La valeur courante de β est une estimation du nombre de processus pouvant avoir crashé depuis $\tau = est_i.set$. Ainsi, p_i attend jusqu'à ce qu'il ait obtenu $|est_i.set| - \beta$ réponses, *i.e.* , l'ensemble des processus vivants à l'instant $est_i.set$, moins ceux qui ont pu crasher depuis (ligne 03).

Du côté pratique, notons que l'obtention de $|est_i.set| - \beta$ réponses peut avoir deux causes : soit de nouvelles réponses sont arrivées, soit β a augmenté car p_i attend depuis trop longtemps.

Nous allons maintenant étudier les propriétés du protocole présenté. Le théorème 1 prouve que chaque crash est inéluctablement détecté, et le théorème 2 donne le sens de $est_i.date$: ces deux théorèmes définissent la propriété de sûreté du protocole. Le théorème 3 porte sur la vivacité.

Théorème 1 *Soit une exécution dans laquelle p_k est un processus défaillant et p_i un processus correct. Il existe un temps après lequel p_k n'appartient pas à $est_i.set$.*

Preuve Soit τ un instant après lequel aucun processus ne reçoit de messages de p_k (ce temps existe puisque p_k, défaillant, n'envoie qu'un nombre fini de messages). Après τ, aucun processus p_j n'inclut p_k dans son $rec\_from_j.set$. Il est possible qu'à l'instant τ, il existe encore des réponses dont l'ensemble contient p_k. Soit alors $\tau' \geq \tau$ l'instant à partir duquel plus aucun processus ne reçoit de réponse transportant un $rec\_from.set$ qui contient p_k. Cet instant existe puisque τ existe et que les messages transitent en un temps fini. D'après les lignes 03 et 08, après τ', aucun processus n'insère p_k dans son $est_i.set$ $\square_{Théorème}$

Théorème 2 *Soit un appel à* estimate() *par p_i. Aucun processus de $est_i.set$ retournés par cet appel n'étaient crashés à l'instant $\tau = est_i.date$.*

Preuve Rappelons qu'aucun processus de l'ensemble $rec\_from_j.set$ d'un processus p_j n'était crashé lorsque p_j a émis la requête correspondante (sinon ce processus n'aurait pas répondu à la requête). Les processus de l'ensemble $rec\_from_j.set$ étaient donc vivants à l'instant $\tau_j = rec\_from_j.date$ (l'instant où p_j a émis la requête).

Considérons maintenant un processus p_i calculant respectivement $est_i.date$ et $est_i.set$ aux lignes 07 et 08. Le théorème provient du fait que (1) $est_i.set$ est l'union des $rec\_from_j.set$ reçus, et (2) $est_i.date$ est la plus petite date de l'ensemble des $rec\_from_j.date$ reçus. $\square_{Théorème}$

Théorème 3 *Chaque invocation de* estimate() *par un processus correct termine.*

Preuve La seule ligne pouvant bloquer un processus p_i est la ligne contenant l'instruction **wait until** (lignes 03-04). Par contradiction, supposons que p_i bloque toujours à cette ligne. D'après la ligne 04, p_i calcule continuellement les nouvelles valeurs de β. Comme $est_i.date$ ne change

pas entre deux évaluations successives de β, mais que les valeurs rendues par les invocations de global_clock() sont croissantes, et d'après les propriétés de $\alpha()$ (elle tend vers $n-1$), le prédicat $|est_i.set| - \beta \leq 1$ devient inéluctablement vrai. Comme les canaux sont sûrs, p_i reçoit au moins sa propre réponse. Ainsi, il existe un instant à partir duquel $|est_i.set| - \beta \leq 1$ est vrai et p_i a reçu sa propre réponse. A cet instant, p_i arrête d'attendre, ce qui donne la contradiction. □*Théorème*

Protocole base sur les horloges locales

```
operation estimate():
1:    broadcast QUERY();
2:    wait until (|est_i.set| − β)) corresponding RESPONSE(rec_from_j, local_date_j, helping_date_j[i])
3:        have been received where β = α(local_clock() − est_i.date) is continuously evaluated;
          % If any, when they arrive, the other corresponding response messages are discarded %
4:    rec_from_i ← the set of processes from which p_i has received RESPONSE() at line 02;
5:    for each p_j ∈ rec_from_i do helping_date_i[j] ← last_date_i[j];
6:                                 last_date_i[j]      ← local_date_j end do;
7:    est_i.date ← min of the helping_date_j[i] received at line 02;
8:    est_i.set  ← ∪ of the rec_from_j sets received at line 02;
9:    return (est_i.set)

background task T: when QUERY() is received from p_j
      do send RESPONSE(rec_from_i, local_clock(), helping_date_i[j]) to p_j end_do
```

FIG. 3.2 – L'opération estimate() (version basée sur des horloges locales non synchronisées)

Nous avons introduit le protocole en utilisant des horloges globales. Nous allons maintenant voir comment s'en passer, afin de ne plus utiliser que des objets « locaux ». En effet, l'horloge, maintenant locale et non synchronisée n'a plus aucune utilité hormis pour le processus l'hébergeant.

Le problème est donc maintenant d'associer pour chaque p_i une date locale τ_i à chaque $est_i.set$ telle que τ_i soit la plus récente possible, et que tous les processus de $est_i.set$ étaient vivants à cette date (en supposant l'existence d'un observateur externe utilisant l'horloge de p_i pour dater tous les évènements du système). Dès qu'un tel temps est déterminé, c'est gagné, p_i peut calculer une estimation du nombre de processus ayant crashé depuis la dernière estimation et ainsi savoir combien de messages attendre.

Pour atteindre cet objectif, nous allons avoir recours aux mêmes variables locales, ainsi que quelques unes supplémentaires :

- est_i reste la même variable, la seule différence notable est que $est_i.date$ fait maintenant référence à une date locale, c'est-à-dire relativement à l'horloge locale de p_i.
- $rec\_from_i.date$ disparaît, on note alors $rec\_from_i.set$ simplement rec_from_i .
- Chaque processus a recours à deux tableaux locaux de taille n : $helping\_date_i[1..n]$ et $last\_date_i[1..n]$. Maintenant, lorsqu'un processus p_j répond à une requête émise par p_i, il envoie la valeur courante de son horloge locale (voir la partie « background task », figure 3.2). Lorsqu'il la reçoit, p_i stocke cette date dans $last\_date_i[j]$ (ligne 05). Rappelons que cette date n'a de sens que pour p_j. Néanmoins, ainsi, p_i est dans la possibilité d'indiquer à p_j la date à laquelle p_j a envoyé sa dernière réponse à p_i.
  Malheureusement, ce mécanisme seul n'est pas suffisant (voir théorème 4) pour garantir la propriété énoncée ci-dessus (tous les processus de $est_i.set$ étaient vivants à l'instant $est_i.date$. En fait, il ne faut pas donner la date de la dernière réponse, mais celle de

l'antépénultième. Celle-ci est conservée dans $helping\_date_i[j]$.

Le code du protocole pour un processus p_i est donné figure 3.2. Globalement, ce code demeure inchangé : seule la partie gestion du temps change. Lorsque p_i répond à p_j, il envoie la valeur courante de son horloge locale (afin que p_j puisse l'aider), la valeur courante de l'ensemble rec_from_i, ainsi que la valeur courante de $helping\_date_i[j]$, afin d'aider p_j dans ses calculs.

De son côté, lorsqu'il reçoit une valeur $helping\_date_i[j]$ d'un processus p_j (ligne 02), p_i l'utilise afin de calculer la date $est_i.date$ qu'il associe à l'ensemble $est_i.set$ (line 07).

Les théorèmes 1, 2 et 3 restent vrais lorsque l'on considère l'algorithme de la figure 3.2. Les preuves des théorèmes 1 et 3 sont presque identiques : l'adaptation est triviale. Ce n'est pas le cas de celle du théorème 2. Nous allons ici présenter une preuve du théorème 2 adaptée au cas des horloges locales.

Théorème 4 *Soit un appel à* estimate() *par p_i. Aucun des processus de $est_i.set$ retournés par cet appel n'était crashé à l'instant $\tau = est_i.date$.*

Preuve Considérons un processus p_k appartenant à l'ensemble rec_from_j que p_i utilise pour définir la nouvelle valeur de $est_i.set$ (ligne 08). Nous montrons que p_k était vivant à l'instant $est_i.date$ exprimé dans l'heure locale à p_i.

La figure 3.3 illustre la situation : cette figure décrit deux requêtes émises par p_j, donnant lieu à deux réponses de p_i qui sont reçues par p_j. Soit τ_i^1 et τ_i^2 la valeur courante de l'horloge locale de p_i lorsqu'il répond aux requêtes.

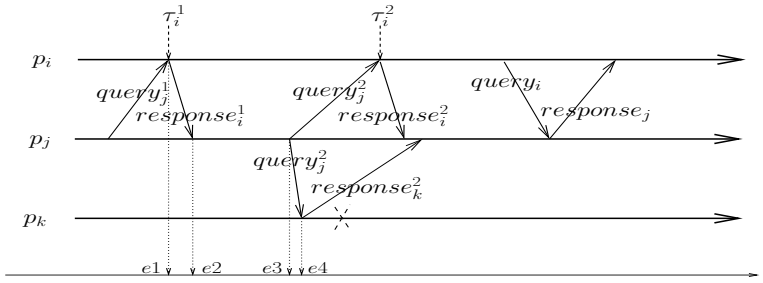


FIG. 3.3 – Utilisation de la relation *happened before*

D'après le protocole, et après le traitement par p_j des réponses des processus p_i et p_k, à savoir les messages $response_i^2$ et $response_k^2$, nous avons :

- $helping\_date_j[i] = \tau_i^1$ et $last\_date_j[i] = \tau_i^2$ (lignes 05-06, exécutées par p_j).
- $p_k \in rec\_from_j$ (ligne 04, exécutée par p_j).

Première observation, p_k peut crasher juste après avoir envoyé sa réponse $response_k^2$ (ce crash est symbolisé par une croix sur la figure). Un tel crash peut arriver avant que p_i ne reçoive le message $query_j^2$, ce qui veut dire que p_k crashe avant que l'horloge de p_i devienne égale à τ_i^2. Cette observation permet de conclure que lorsqu'il reçoit de p_j le message $response_j$ (avec un rec_from_j tel que $p_k \in rec\_from_j$), p_i ne peut pas conclure que p_k était vivant à l'instant τ_i^2.

Le message $response_j$ envoyé par p_j transporte la date $helping\_date_j[i] = \tau_i^1$, et p_i utilise cette date (qui lui est locale) lors de son calcul de $est_i.date$ (ligne 07). Nous avons donc $est_i.date \leq \tau_i^1$. Pour montrer que p_k était vivant lorsque l'horloge de p_i valait τ_i^1, nous utilisons la relation *happened before* le Lamport [Lam78].

- p_j envoie sa deuxième requête ($query_j^2$) *après* avoir traité la réponse $response_i^1$ de p_i.
- p_k envoie $response_k^2$ *après* avoir reçu la requête correspondante ($query_j^2$).

– Donc d'après la relation *happened before*, l'évènement $e1$ *précède* l'évènement $e4$, ce qui permet de conclure que p_k était vivant lorsque l'horloge locale de p_i valait τ_i^1, ce qui prouve le théorème.

□*Théorème*

Lorsque n est inconnu

Imaginons un système procurant aux noeuds une fonction de diffusion permettant de joindre tous les noeuds du système, mais où le nombre total d'entités, n, est inconnu.

Il est intéressant de noter que dans ce contexte, les deux protocoles présentés précédemment ne nécessitent pas la connaissance de n. Bien sûr, il faudrait pour cela remplacer les tableaux de dimension n par des listes, ou toute autre structure de données à taille non bornée. Cela veut dire que le code des deux protocoles est indépendant de la taille du système.

3.1.5 Évaluation expérimentale

Comme indiqué au début de cette section, il n'y a dans un système asynchrone pur *aucun* moyen de différencier un processus crashé d'un processus très lent. Ceci est en effet impossible puisque cela requiert la définition d'une limite temporelle au-delà de laquelle le processus est considéré comme crashé, et puisque l'asynchronie peut faire arriver tous les messages à l'instant suivant cette limite. Dans le modèle que nous venons de présenter, les processus ont aussi

– accès à une horloge locale pour mesurer les durées
– accès à une fonction $\alpha()$ prédéfinie.

Ce n'est pas suffisant non plus pour permettre de différentier parmi les processus les morts des lents. Le détecteur de défaillances (l'oracle) permettant de faire cette distinction s'appelle le détecteur parfait [CT96, Ray05], et les hypothèses nécessaires à son implémentation sont bien plus fortes.

Loin d'être inutile, notre protocole a toutes les propriétés énoncées dans les théorèmes 1-4 et rend des *estimations* du nombre de processus vivants. La question maintenant est de savoir quel est le niveau de précision de ces estimations. D'un façon plus poussée même, il s'agit de savoir qu'est-ce qui rend les estimations précises. Nous allons répondre à cette question par des expérimentations. Les simulations utilisent le protocole basé sur les horloges globales 3.1.

Le modèle de simulation

La réponse à la question précédente dépend de beaucoup de paramètres, dont la façon dont s'enchaînent les crashs, la distribution des délais et, plus généralement, la structure sous-jacente du réseau de communication.

Tout d'abord, nous utilisons un simulateur à temps discret. Nous considérons des rondes durant lesquelles chaque processus fait un appel à la procédure estimate(). La ronde se termine lorsque tous les processus ont terminé l'exécution d'estimate() (elle a donc une longueur variable du point de vue des horloges locales des processus). Ce choix est une simplification et à ce titre il cache un problème qu'il est possible de rencontrer dans un système totalement asynchrone. En effet : imaginons un système comportant un processus p_l n'appelant que rarement estimate(). Les autres processus lors de la datation de leur vue, c'est-à-dire lors du calcul d'une date $est_i.date$ associée au $est_i.set$, vont devoir prendre la date de la vue de p_l qui est très ancienne. Cela peut mener à des pertes de performances pour le système entier. Cependant, il est facile de corriger se problème en exploitant de manière plus précise les vues reçues, comme souligné dans la section 3.1.4. La

vision en rondes permet une approche beaucoup plus simple de la simulation et la compréhension des résultats en est grandement simplifiée.

Considérons le déroulement des crashs. Nous le verrons par la suite, le protocole présenté est très robuste en situation « stable », à savoir lorsque tous les processus connaissent une bonne partie des acteurs du système. Notre métrique principale sera la quantité de fausses suspicions du système. Nous allons supposer le pire cas, c'est-à-dire lorsqu'il n'y a pas de crashs. Ainsi, toute suspicion est une fausse suspicion. Nous supposons un système composé de $n = 100$ processus. Notons que supposer qu'un processus crashe reviendrait, après ce crash, à simuler le protocole avec 99 entités puisque le protocole est indépendant du nombre d'entités. Pour la fonction $\alpha()$, on considère arbitrairement un crash par unité de temps.

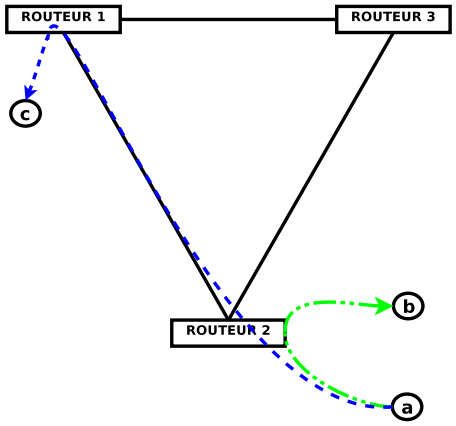


FIG. 3.4 – Modélisation du réseau sous-jacent

Dans cette première simulation, nous allons nous contenter d'une modélisation simple du système :

- A chaque processus, on attribue une position aléatoire (loi uniforme) dans un espace à 2 dimensions
- On dispose r routeurs dans l'espace
- On rattache chaque processus à son routeur le plus proche.
- A chaque ronde, le simulateur calcule les temps de transit des messages. Soit t^i_{ab} le temps de transfert d'un message de p_a à p_b à la ronde i. On a $t^i_{ab} = td_{ab} + \mathcal{N}^i$, ou $\mathcal{N}^i$ est la composante aléatoire du temps de transit du message (loi Normale), qui varie d'une ronde à l'autre, et td_{ab} un temps « statique » proportionnel à la distance entre p_a et p_b. Soient R_{p_a} et R_{p_b} les routeurs de p_a et p_b, respectivement. On a

$$td_{ab} = d(p_a, R_{p_a}) + c \times d(R_{p_a}, R_{p_b}) + d(p_b, R_{p_b}),$$

  où $d(i, j)$ désigne la distance euclidienne entre les points i et j, et c une variable permettant de moduler le coût du transit entre routeurs. Notons que cette formule est aussi vraie dans le cas où $R_{p_a} = R_{p_b}$: les processus ne communiquent jamais directement.

Ce modèle est illustré figure 3.4 : p_a et p_b sont rattachés au routeur $ROUTEUR_1$ et p_c est rattaché à $ROUTEUR_2$. Les flèches pointillées désignent le trajet de messages de p_a à p_b et p_c. Dans les simulations présentées, nous avons $r = 3$ et $c = 3$, et la distance entre deux routeurs différents est constante. Un nombre plus important de routeurs n'impacte que peu les résultats, et considérer 2 routeurs constitue un cas un peu particulier qui mène facilement au partitionnement du réseau, que nous décrirons par la suite. Faire varier c permet d'éloigner ou de rapprocher les routeurs. Les simulations présentées dans la section suivante étudient l'impact de ces variations. Dans cette partie nous nous contentons de simuler un temps de transit 3 fois plus long.

Quelle est la précision du protocole ?

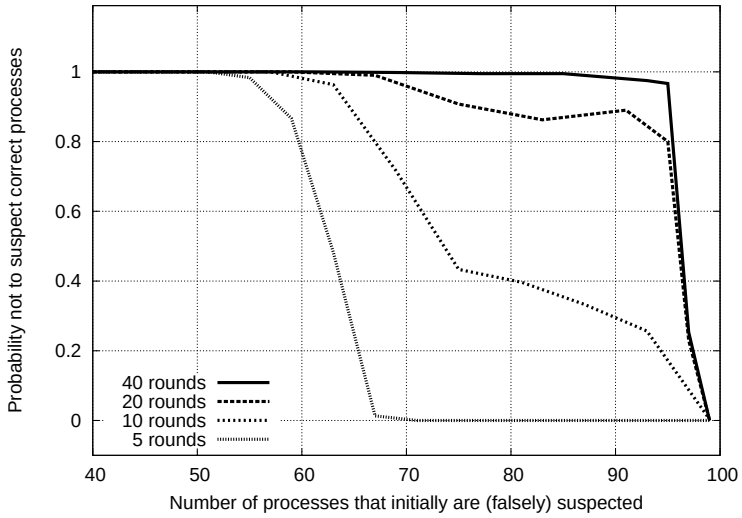


FIG. 3.5 – Probabilité de ne pas suspecter de processus correct

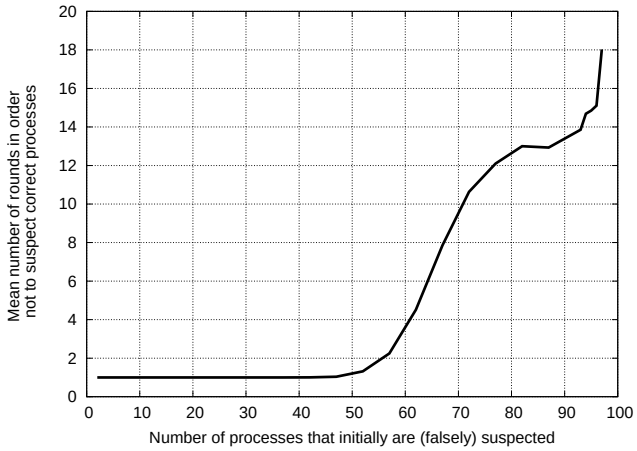


FIG. 3.6 – Résultat de simulation : A quelle vitesse le protocole procure-t-il une estimation précise ? (nombre de rondes nécessaires au rétablissement en fonction du nombre de fausses suspicions initiales)

Nous avons vu (théorème 1) qu'un processus crashé est inéluctablement suspecté. En se plaçant du point de vu d'un utilisateur, nous définissons la *précision* du protocole comme la probabilité de produire des réponses complètes, c'est-à-dire contenant tous les processus vivants. Nous pouvons voir la précision comme la mesure de la couverture ([Pow92]) de l'assertion « aucun ensemble rendu n'omet de processus correct ».

Les expériences que nous présentons ici étudient cette propriété de précision. Afin de « stresser » le protocole, nous introduisons au début de l'expérience de fausses suspicions. Il est possible d'imaginer l'expérience démarrant après une période de forte asynchronie du système : chaque processus suspecte à tort un certain nombre d'autres processus. Nous regardons ensuite comment sont résorbées ces fausses suspicions.

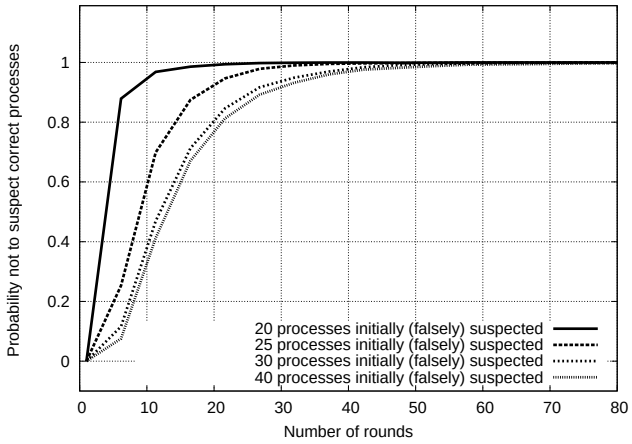


FIG. 3.7 – Résultat de simulation : A quelle vitesse le protocole procure-t-il une estimation précise ? (probabilité de rétablissement en fonction du temps)

Les résultats de cette expérience sont présentés figure 3.5 : cette figure présente la probabilité que tous les processus aient une vue précise du système, après 5, 10, 20 et 40 rondes. Cette figure montre la stabilité du protocole : lorsque le nombre de fausses suspicions introduites ne dépasse pas la moitié du nombre de processus du système, celles-ci sont résorbées après moins de 5 rondes dans la quasi-totalité des cas. Plus encore, il faut introduire plus de 60% de fausses suspicions pour voir la probabilité de se rétablir en moins de 5 rondes chuter.

Ainsi, la figure 3.5 montre que la précision du protocole ne se rétablit pas dans un nombre raisonnable de rondes que lorsque les processus suspectent à tort plus de 95% du système.

Quelle est la vitesse de convergence du protocole ?

Une deuxième question importante est celle de la vitesse de convergence du protocole. Nous avons déjà vu l'impact du nombre de rondes attendues au point précédent : attendre plus augmente les chances d'obtenir une vue précise. Maintenant nous retournons la question : à quelle vitesse obtient-on une vue précise ? Ce point est traité par les figures 3.1.5 et 3.1.5.

La figure 3.1.5 nous donne le nombre de rondes à attendre en moyenne en fonction du nombre de fausses suspicions à dissiper. Ainsi, dans un système où tous les processus suspectent faussement 70% des processus, il faudra attendre en moyenne 10 rondes d'échange avant que plus aucun processus n'ait de fausse suspicion. Il est intéressant de noter que cette courbe exhibe un effet de percolation (l'étage de $x = 50$ à $x = 80$) dont le milieu (env. $x = 65$) se trouve être le nombre moyen de processus par routeur. Nous verrons en effet par la suite qu'il est plus facile de suspecter les processus appartenant à d'autres routeurs qu'au sien. Encore un fois, on peut observer qu'un nombre raisonnable de fausses suspicions est corrigé très rapidement et qu'après 95% de fausses suspicions, la correction est difficile.

La figure 3.1.5 présente un autre angle de vue : dans un système de 50 processus, nous introduisons 20, 25, 30 et 40 fausse suspicions. Après chaque ronde, nous regardons si l'estimation est précise pour tous les processus. Répétée plusieurs fois, cette expérience permet de représenter la probabilité de rétablissement en fonction du nombre de rondes. Premier constat, à nouveau la stabilité : toutes les courbes converges. Pour les deux cas où les suspicions n'excèdent pas la majorité des processus, cette convergence est même très rapide (moins de 10 rondes dans la moitié des cas). Cette figure résume ainsi les précédentes : 1) le protocole converge, 2) généralement très rapidement.

3.1.6 Impact du réseau sous-jacent

Dans cette section, nous étudions l'impact de la structure sous-jacente, c'est-à-dire l'impact du réseau utilisé pour interconnecter nos processus. En effet, pour communiquer, dans la majorité des réseaux utilisés, les processus vont avoir recours à des intermédiaires qui vont se charger d'acheminer les paquets d'information à destination.

Nous avons présenté dans les sections précédentes un protocole qui a la particularité de « s'auto-alimenter », dans le sens où les résultats d'une ronde vont conditionner la quantité d'information obtenue à la ronde suivante, donc le résultat de la ronde suivante, ainsi de suite. Dans les cas où le protocole converge, ce cercle d'information est vertueux : chaque échange apporte de plus en plus d'information, jusqu'à la découverte de la totalité du système. Dans les cas où le protocole ne converge pas, ce cercle est vicieux : les processus n'obtiennent pas assez d'information pour attendre plus à la ronde suivante, qui du coup n'apportera pas plus d'information. A terme, chaque processus reste avec une vision partielle du système : le système est partitionné.

Penchons-nous sur l'ensemble des réponses collectées à une ronde par un processus donné. Il est important d'observer que toutes les réponses n'ont pas la même valeur, ou le même intérêt : certaines réponses apportent des informations nouvelles et d'autres n'apportent presque rien au processus. Soit un processus collectant 3 messages, chaque message contenant un ensemble de 3 identités de processus. Si ces 3 ensembles d'identités sont identiques, le processus n'attendra à nouveau que 3 réponses à la ronde suivante : cette ronde n'a pas apporté d'information. Si ces 3 ensembles sont différents, le processus peut attendre 9 réponses à la ronde suivante, cette ronde a été utile. Ceci illustre l'importance déroulement des échanges de messages.

Les échanges de messages, et donc d'information, sont influencés par le système : les messages pris en compte sont les messages arrivant le plus vite. La vitesse de transit d'un message dépend de la façon dont sont connectés les processus entre-eux. Le réseau sous-jacent, qui est par définition la représentation de ces connections, a donc un impact non négligeable sur $1)$ la vitesse de convergence du protocole, et même $2)$ dans certains cas sur la convergence même du protocole.

Dans les sections précédentes, ces communications sous-jacentes étaient représentées par un ensemble de routeurs interconnectés. Deux processus connectés à des routeurs différents voyaient leurs messages souffrir d'un retard (paramétré par c). La courbe de la figure 3.1.5 présente les résultats obtenus dans ce modèle.

Dans cette section, nous étudions l'impact de la distance entre les routeurs sur la convergence du protocole. La figure 3.8 présente différentes courbes obtenues à partir du protocole présenté section 3.1.4. Chaque courbe est le résultat de plusieurs exécutions avec un délai inter-routeurs différent. Les différences entre les courbes montrent l'impact non négligeable de ce paramètre : plus le délai est haut, plus la convergence est rapide. Cependant, avec un délai trop important, le temps de convergence au delà de 70% de fausses suspicions augmente exponentiellement et le protocole converge rarement.

L'explication vient de l'intérêt des réponses collectées par les processus à chaque ronde.

- Avec un petit délai inter-routeur, les messages ont à peu près tous la même chance d'arriver à chaque processus (le hasard a un grand poids). Ainsi chaque processus a une vue (c'est-à-dire l'ensemble des processus qu'il connaît) plutôt aléatoire du système. Il a donc de grandes chances d'apporter *un peu* d'information nouvelle aux processus auxquels il répond.
- Au contraire, avec un grand délai inter-routeurs, les échanges « intra-routeur » sont grandement favorisés par rapport aux échanges inter-routeur. Les processus se découvrent rapidement dans un routeur, et les message ont peu de chance de passer entre les routeurs. Ceci a deux effets : lorsqu'un message inter-routeur passe, il apporte énormément d'information nouvelle au processus qui le reçoit, ce qui permet une convergence rapide. Par contre, lorsque aucun message inter-routeur ne passe, le système reste partitionné, chaque processus

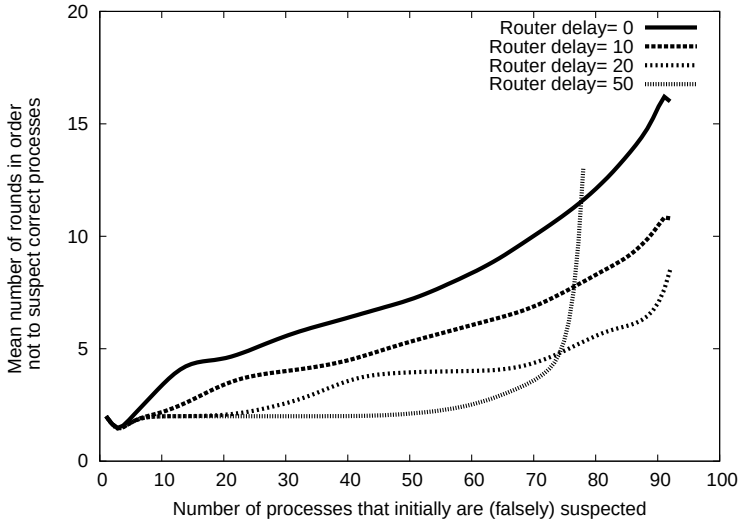


FIG. 3.8 – Temps de convergence pour différents délais inter-routeurs

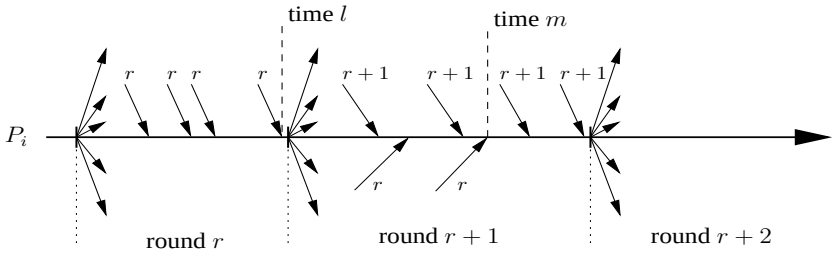


FIG. 3.9 – Temps d'arrivée des messages

connaissant les processus de son routeur et ignorant les autres.

Sur la figure 3.8, il est intéressant d'observer que la courbe avec un délai de 20 unités de temps constitue le meilleur compromis : une convergence rapide dans tous les cas. Malheureusement, le délai entre routeurs n'est pas un paramètre contrôlable par l'administrateur du système.

Ce qu'il faut donc retenir ici c'est l'influence de la structure sous-jacente sur les schémas d'échanges de messages et, par cet intermédiaire, sur les performances de ce protocole réparti. En retardant certains messages relativement à d'autres, la structure sous-jacente modifie donc la perception qu'ont les processus du système. Le problème ici est que l'influence n'est pas un paramètre que l'administrateur du système peut contrôler.

3.1.7 Compenser les effets néfastes de la couche sous-jacente

Nous présentons maintenant une technique pour gommer l'influence de la couche de communications. Plus encore, ces modifications permettent de se rapprocher de la situation où l'éloignement des routeurs (*i.e.* l'ordre d'arrivée des messages) assure une convergence rapide dans la majorité des cas.

Rappelons que les processus n'ont aucune information a priori sur la structure du réseau de communication. L'idée ici est de forcer chaque processus à attendre un certain laps de temps (déterminé localement selon une méthode que nous allons détailler ensuite) avant d'envoyer ses messages au départ de chaque ronde. Bien sûr, si tous les processus utilisent le même délai, aucun changement ne sera observé. Le principe est ici de favoriser un petit nombre de messages échangés

par des processus distants afin d'établir des échanges d'information inter-routeurs.

A cette fin, nous associons deux dates locales aux arrivées de messages pour chaque processus. Notons qu'un message envoyé à la ronde r peut être reçu à la ronde r'. Les dates sont définies ainsi :

- ℓ_r^r est la date d'arrivée (mesurée par l'horloge locale de p_i) du dernier message de la ronde r reçu par p_i alors qu'il exécutait cette ronde r (*i.e.* la date d'arrivée du dernier message pris en compte à la ronde r)
- Il est possible que des messages envoyés à p_i à la ronde r lui arrivent alors qu'il a déjà entamé la ronde $r+1$. Soit ℓ_r^{r+1} la date d'arrivée du dernier message de la ronde r qui est reçu par p_i à la ronde $r+1$ (cela correspond à l'instant m sur la figure 3.9). Notons qu'il est possible qu'aucun tel message n'arrive à la ronde $r+1$. On a alors $\ell_r^{r+1} = \ell_r^r$.

La paire (ℓ_r^r, ℓ_r^{r+1}) calculée par p_i peut être utilisée par p_i à la ronde $r+2$. Plus précisément, p_i retarde l'émission de ses réponses d'un délai uniformément tiré dans l'ensemble $[0, \ell_r^{r+1} - \ell_r^r]$. L'idée derrière ce retard est de favoriser l'échange de messages entre processus de routeurs différents. Ainsi lorsque p_i rajoute du délai, il donne plus de chance aux messages émis par des processus distant d'être pris en compte par des processus proches de p_i (qui sinon auraient pris le message de p_i). Cela permet d'améliorer la précision du protocole malgré l'hétérogénéité du réseau de communication sous-jacentes. Notons enfin que lorsque p_i ne rate pas de messages, il n'ajoute pas de délai, ce qui rend l'amélioration transparente en situation « normale ».

La figure 3.10 illustre les bénéfices de l'utilisation des dates ℓ_r^r et ℓ_r^{r+1}. Ces courbes ont été obtenues dans la même configuration que la figure 3.8 mis à part que la figure 3.10 utilise l'amélioration que nous venons de décrire. Les différences entre ces deux courbes sont donc uniquement dues au bénéfice de l'utilisation des dates : la convergence est plus rapide dans toutes les situations. De plus, dans le cas d'un délai élevé (50), la convergence est maintenant assurée au delà de 70% de fausse suspicions, seuil ou le système reste partitionné dans la figure 3.8.

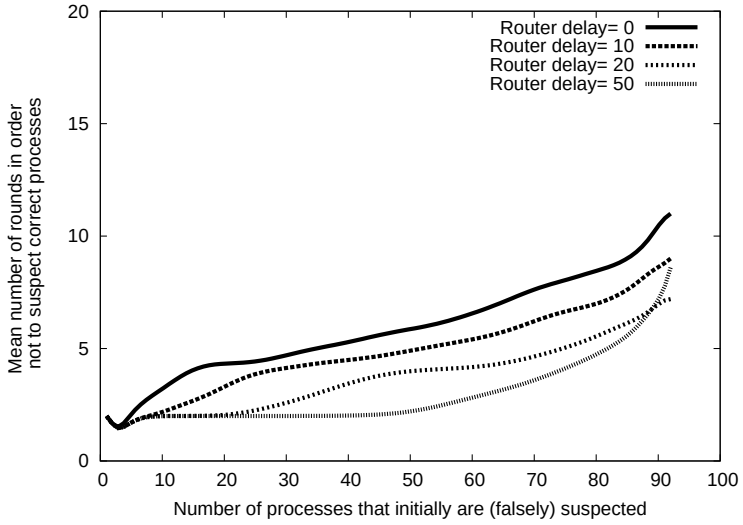


FIG. 3.10 – Temps de convergence pour différents délais inter-routeurs — avec délai d'émission

Il est possible d'analyser ces résultats avec un point de vue petit monde [Kle00] : les processus d'un même cluster se découvrent facilement et mutuellement ; ils échangent facilement de l'information et peuvent être considérés comme des *voisins locaux*. Comme décrit précédemment, favoriser l'échange d'information entre processus de différents clusters améliore beaucoup la vitesse de convergence, puisque le message qui passe ainsi apporte beaucoup d'information nouvelle (puisqu'elle concerne un cluster ignoré jusqu'alors). Ceci peut être vu comme ajouter des liens longs

(ou des raccourcis, selon la terminologie consacrée [Kle00]), en opposition aux voisins locaux. De tels raccourcis sont reconnus très utiles pour le routage. On peut ainsi voir l'amélioration que nous proposons comme une simulation de ces liens longs.

Enfin, ajoutons qu'il existe dans ces résultats un petit artefact qui mériterait de plus amples investigations : notre décompte du temps est basé sur les rondes, et nous jugeons la vitesse de convergence au nombre de rondes nécessaires à rétablir une vue parfaite. En introduisant des délais dans les attentes, nous augmentons la durée moyenne de la ronde et ce de façon invisible pour notre métrique. Il serait intéressant d'observer le temps total mis pour la convergence, ou au moins l'évolution de la durée moyenne de la ronde.

3.2 Centralité du second ordre : calcul réparti de l'importance des noeuds

Dans la section précédente, nous avons présenté un protocole et constaté l'influence de la structure des communications sous-jacentes dans le contexte des systèmes asynchrones. Les processus étaient discriminés selon leur rapidité à échanger des messages.

Dans cette section, nous utilisons un modèle synchrone. Dans ce modèle, il existe (bien qu'elle ne soit pas explicitement définie) une borne haute Δ_{max} sur les temps de transfert des messages. Les moyens de communication sont supposés fiables, et ainsi deux processus échangent de l'information en Δ_{max} au maximum. Les processus ne sont plus discriminés selon leur rapidité à échanger des messages : dans cette section, ils ont discriminés selon leurs interlocuteurs.

Ici en effet, nous nous plaçons dans le contexte des réseaux large échelle. Et comme indiqué dans l'introduction, supposer qu'un processus connaît tous les autres participants du réseau est une hypothèse très (trop) forte. Nous supposerons donc que chaque processus ne communique qu'avec un petit nombre de processus du système : ses voisins. La structure sous-jacente des communications devient alors plus explicite : il s'agit du graphe d'interaction des processus.

L'objectif de cette section va être de procurer aux processus une métrique capturant l'importance du rôle qu'ils jouent dans cette structure de communication. Cette métrique s'appelle la *centralité du second ordre*. Nous allons ici présenter la centralité du second ordre, illustrer son intérêt à l'aide de simulations et enfin comparer cette centralité aux autres centralités existantes.

3.2.1 Intérêt des mesures de centralité

Les réseaux pair-à-pair, les réseaux organisationnels ou les réseaux sociaux, ou encore les grilles ou les réseaux de capteurs, présentent de plus en plus d'énormes structures d'interaction. On parle de réseau large échelle. Ces structures d'interaction sont des graphes dont la taille rend la manipulation et l'analyse difficile [AB02, LM06]. De plus, l'accès à la topologie complète du graphe n'est pas toujours possible (*p.ex.* réseaux p2p cryptés) ; un autre cas de figure est l'industrie qui préfère exploiter la puissance de calcul des machines de ses clients plutôt que d'investir dans des fermes de calcul coûteuses en entretien. Il est alors nécessaire de concevoir des méthodes réparties d'analyse de ces graphes.

L'analyse des graphes statiques, particulièrement des réseaux sociaux, est un sujet étudié depuis longtemps par les physiciens et les sociologues. Ces travaux ont donné naissance à la notion de *centralité* [Bar04, BP07, Fre77, New05], qui capture l'importance de chaque noeud dans un graphe d'interactions. Un système réparti peut grandement bénéficier de telles informations, par exemple pour éviter que la topologie ne dépende que de quelques noeuds, ou encore pour prévoir les partitionnements, ou répartir la charge applicative. Comme un noeud peut être considéré comme important à plusieurs titres, il existe différents types de centralité. La section 3.2.3 présente les centralités les plus courantes.

Malheureusement, la majeure partie de ces centralités sont conçues pour un calcul centralisé (analyse *offline*) et sont ainsi difficilement adaptables aux contextes répartis. Partant de ce constat, cette section comprend plusieurs contributions :

1. elle introduit une nouvelle forme de centralité, c'est-à-dire une nouvelle métrique permettant de capturer à la fois l'importance de chaque noeud donné par rapport à la santé de la topologie[1] dans laquelle il se trouve, et aussi la santé globale de la topologie. Nous montrerons ainsi comment exploiter cette centralité pour identifier les noeuds dits *critiques* (dont le rôle est très important), ainsi que les clusters.
2. elle présente un algorithme léger et complètement réparti permettant à chaque noeud de connaître sa centralité. La force de cet algorithme repose dans sa simplicité : il repose sur une marche aléatoire (définie section 3.2.2) arpentant le réseau de noeud en noeud. Chaque noeud enregistre les temps de retour (c'est-à-dire le temps écoulé entre deux visites consécutives de la marche). L'écart type de ces temps de retour représente l'importance du noeud.
3. Nous montrons comment cet algorithme peut être utilisé pour créer des signatures de graphes et ainsi donner d'importantes informations sur les propriétés globales d'un graphe.
4. A travers des simulations, nous montrons que l'algorithme est utilisable dans des contextes pratiques, et nous comparons les résultats obtenus aux résultats obtenus à l'aide des autres algorithmes de centralités.

3.2.2 Modèle et principes du protocole

Nous nous intéressons ici à un ensemble d'entités (que nous appelons noeuds) chacune intéragissant avec une petite partie des autres entités du système (que nous avons défini comme ses voisins. Comme mentionné en introduction, ce modèle trouve son fondement en informatique dans les grands réseaux pair-à-pair, où il est impossible de supposer que chaque entité connaisse toutes les autres entités du système. Les réseaux sociaux forment aussi de très bon exemples de tels systèmes : on peut citer le cas des échanges de courriers électroniques dans une université [GDG+03], le réseau formé des collaborations des scientifiques dans un domaine donné [New06], ou encore le réseau des partenaires sexuels [LEA+01].

Formellement, ces interactions forment un graphe. Nous considérons ainsi un réseau arbitraire, représenté sous la forme d'un graphe non dirigé $\mathcal{G} = (V, E)$, avec n sommets (qui sont les noeuds, ou encore les entités) et m arcs. Ces arcs capturent la relation de collaboration, quelle qu'elle soit : $(i, j) \in E \Leftrightarrow$ les entités i et j collaborent (ont échangé un mail ou ont cosigné un article pour reprendre les exemple du paragraphe précédent). Pour un noeud $i \in V$, Γ_i est le voisinage de i dans $\mathcal{G}$ (c'est-à-dire les sommets qui ont collaboré avec i), et d_i son degré, c'est-à-dire la taille de Γ_i. Nous supposons le graphe connecté : c'est crucial pour éviter d'obtenir un résultat partiel sur la composante connexe d'où est partie l'unique marche.

Nous utilisons une *marche aléatoire* (ou *random walk*) sur le graphe $\mathcal{G}$, c'est-à-dire un message (on parle aussi d'un processus, mais au sens mathématique) progressant à travers le graphe de i à un autre noeud choisi uniformément dans le voisinage Γ_i de i. Nous considérons cette marche *permanente* c'est-à-dire qu'elle ne possède aucune condition d'arrêt ; c'est d'ailleurs une grande différence avec la majorité des marches aléatoires du domaine informatique (*e.g.* le sampling [GKMM07] ou la recherche [LCC+02]). On suppose que la marche est lancée par un noeud

[1] Une topologie en bonne santé s'entend ici comme issue d'un graphe non problématique, c'est-à-dire où les chances de partitionnement sont faibles, où le diamètre est raisonnable comparé à la taille du graphe et où les noeuds ont des importances comparables

arbitraire dans le réseau. Nous supposons aussi que le graphe est statique, c'est-à-dire que la topologie ne change pas au cours de l'exécution de l'algorithme. Enfin, on suppose que cette marche n'est jamais perdue (c'est-à-dire que le message relayé de proche en proche n'est jamais perdu).

Notre algorithme repose sur l'exploitation des temps de retour, c'est-à-dire sur l'exploitation de la série des temps mis entre deux visites consécutives de la marche aléatoire au même noeud. Ici, le temps peut être défini soit de manière absolue, c'est-à-dire à l'aide d'une horloge disponible sur chaque noeud, soit simplement comme le nombre de pas effectués par la marche aléatoire (qui transporte alors cette information). Notons que dans le cas d'un temps absolu, les horloges présentes sur chaque noeud n'ont pas besoin d'être synchronisées puisque nous sommes juste intéressés par les variations locales des temps de retour.

Ajoutons enfin que nous n'avons besoin que d'une unique marche aléatoire pour la totalité du système, par opposition à une marche que chaque noeud du système lance afin de satisfaire ses propres besoins.

3.2.3 Travaux connexes

Dans cette section, nous allons présenter les différentes approches permettant aussi de caractériser globalement les graphes sur lesquelles elles sont employées. Ensuite nous verrons différentes notions de centralités employées pour obtenir l'importance des noeuds dans un graphe donné. Enfin, nous détaillerons la centralité d'intermédiarité basée sur les marches aléatoires (*random walk betweenness centrality*), car c'est l'approche la plus proche des travaux de cette section.

Niveau "Macro" : caractérisation globale des graphes

La connectivité des graphes est caractérisée par les notions de *spectral gap*, notée λ_2 et *conductance*, notée Φ. Le spectral gap est la plus petite valeur propre positive de la matrice laplacienne de $\mathcal{G}$, notée L ($L = D - A$, avec D la matrice diagonale des degrés et A la matrice d'adjacence de G, en classant les valeurs propres selon : $0 < \lambda_2 \leq \ldots \leq \lambda_n$). La conductance Φ est définie ainsi :

$$\Phi := \inf_{C:|C|\leq N/2} \frac{E(C,\overline{C})}{|C|},$$

où $E(C,\overline{C})$ est le nombre d'arcs dans $\mathcal{G}$ entre l'ensemble de noeuds C et son $\overline{C}$ qui est son complémentaire dans $\mathcal{G}$ (voir *e.g.* [Moh97]). Intuitivement, cela revient à chercher la coupe (*i.e.* la séparation de V en deux partitions) la plus petite (en nombre d'arcs tranchés) qui déconnecte le maximum de noeuds. Enfin, λ_2 et Φ sont liés par l'inégalité de Cheeger [Moh97] :

$$\lambda_2 \geq \frac{\Phi^2}{2\Delta(\mathcal{G})},$$

avec $\Delta(\mathcal{G})$ le degré maximal des noeuds du graphe.

Plus les valeurs de λ_2 et Φ sont faibles, plus la probabilité que les algorithmes utilisant ce graphe se comportent mal est forte. Un exemple particulièrement intéressant est celui du *temps de mixage* d'une marche aléatoire. C'est le nombre de sauts nécessaires pour qu'une marche aléatoire ait une probabilité uniforme de se trouver sur n'importe quel noeud du graphe (c'est-à-dire indépendamment de son point de départ). Ce temps de mixage est relié à la conductance (voir *e.g.* [Lov98]). En résumé, la conductance et le spectral gap capturent la présence de goulots d'étranglement et de problèmes de connectivité à l'échelle de tout le graphe, c'est-à-dire sans pointer spécifiquement quelques noeuds.

Ce sont donc de bons indicateurs de la robustesse de la structure sous-jacente (à savoir le graphe), dont le lien avec les performance des applications utilisant ce graphe est connu et démontré. Malheureusement, ces indicateurs nécessitent, pour être calculés, la connaissance de tout le graphe. Leur calcul est de plus souvent complexe.

Niveau "micro" : influence des individus dans le graphe

Au delà de l'état global d'un graphe, il est intéressant de connaître l'importance de noeuds par rapport à une topologie donnée, ainsi que l'impact de chaque noeud sur la connectivité globale du graphe. A titre d'exemple, si l'on prend un graphe ayant la forme d'un arbre, un noeud feuille a beaucoup moins d'importance que le noeud racine. Cet importance est reflétée par les indices de centralité.

Malheureusement, il n'existe pas de définition formelle d'un noeud *important*. Cette importance peut prendre plusieurs formes et c'est ainsi qu'il existe plusieurs centralités différentes. Nous allons ici détailler les plus connues :

Degrés La forme la plus simple de centralité est la centralité des degrés. Elle traduit l'idée qu'un noeud important est un noeud possédant beaucoup de voisins. On a donc $C_d(i) = d(i)$, la centralité du noeud i. Cette centralité a l'intérêt d'être très facilement calculable. De plus, Albert *et al.* montrent [AB02] que les réseaux sociaux exhibent souvent une distribution des degrés en loi de puissance : quelques individus ont des degrés largement supérieurs à la majorité de leurs congénères, ce sont des individus importants pour la connexité et la navigabilité du graphe.

Proximité Cette centralité traduit l'idée que les noeuds importants sont des noeuds proches de tous les autres. La centralité d'un noeud est ainsi l'inverse de sa distance moyenne à tous les autres noeuds : $C_c(i) = \frac{1}{\Sigma_{j \in V} d(i,j)}$, ou $d(i,j)$ est la distance, en nombre de sauts, entre les noeuds i et j

Excentricité L'eccentricité [HH95] est proche de la proximité : au lieu d'utiliser la distance moyenne à tous les noeuds d'un graphe, on prend la distance maximale. L'idée est qu'un noeud est important s'il est au "centre du graphe". Autrement dit, les noeuds les plus centraux seront avec cette mesure les noeuds situés au milieux des diamètres. On a $C_e(i) = \frac{1}{max_{j \in V} d(i,j)}$

Vecteur propre (*Eigenvector*) Introduite par Bonacich [Bon72], l'intuition de cette centralité est de dire qu'un noeud important est un noeud connecté à d'autres noeuds importants. Cette définition amène la recherche d'un point fixe, d'où l'utilisation des vecteurs propres. On pose $C_\lambda(i) = \frac{1}{\lambda_n} \Sigma_{j \in \Gamma(i)} C_\lambda(j) = \Sigma_{j=1}^{n} A_{i,j} C_\lambda(j)$, avec A toujours la matrice d'adjacence de $\mathcal{G}$. Vectoriellement, si on définit c_λ comme le vecteur contenant les $C_\lambda(i)$, on a $c_\lambda = \frac{1}{\lambda_n} A c_\lambda$, d'où $\lambda_n c_\lambda = A c_\lambda$: c_λ est le vecteur propre associé à λ_n, la plus grande valeur propre de A. Le célèbre algorithme *pagerank* de Google est une variante de cette centralité [PBMW98].

Intermédiarité La centralité d'intermédiarité fait partie de la famille des centralités basées sur les chemins. Elle est une centralité majeure et a fait l'objet de nombreuses études [Fre77, Bar04, Bra01, GSS08]. Elle consiste à calculer pour chaque noeud la proportion de plus courts chemins sur lequel il réside. Formellement, on pose $C_b(i) = \Sigma_{j \neq k \neq i} \frac{\sigma_{jk}(i)}{\sigma_{jk}}$, avec $\sigma_{jk}(i)$ étant le nombre de plus courts chemins d'un noeud j à un noeud k passant par i, et σ_{jk} le nombre total de plus courts chemins de j à k. L'algorithme original nécessite $\Omega(n^3)$ étapes pour obtenir le résultat complet, mais Brandes [Bra01] propose un algorithme en $O(mn)$ étapes, m étant le nombre d'arêtes du graphe. De récentes études expérimentales [BP07, GSS08] proposent des approximations de cette centralité à des fins pratiques.

Intermédiarité basée sur les marches aléatoires Malgré l'importance de la mesure d'intermédiarité classique présentée juste au dessus, celle-ci possède quelques imperfections relevées par Newmann [New05] : le problème vient du fait qu'elle ne considère que les plus courts chemins. Ainsi, considérons le graphe représenté figure 3.11 : il représente deux clusters (matérialisés par les ronds pointillés), connectés par quelques noeuds seulement (a, b et c). Dans ce contexte, c obtient une très faible centralité d'intermédiarité (puisqu'il n'est sur aucun plus court chemin), malgré son importance vitale pour la connexité des deux clusters (il offre une redondance du lien $a - b$).

De tels noeuds, oubliés des plus courts chemins, peuvent être d'un importance vitale pour la robustesse du réseau, ou pour la répartition de charge, ou pour capturer des flux d'informations n'empruntant pas forcément les plus courts chemins. C'est afin de réparer ces lacunes que Newman introduit [New05] la centralité d'intermédiarité basée sur les marches aléatoires : une marche aléatoire prend en compte tous les chemins, pas les plus courts seulement.

Le principe de calcul est simple : chaque noeud s lance une marche aléatoire vers tous les autres noeuds t du graphe. Chaque noeud compte le nombre de telles marches relayées, et le résultat est ensuite divisé par le nombre total de paires (s, t). Une heuristique est aussi ajoutée pour éviter de compter plusieurs fois le passage d'une marche aléatoire dans le cas où celle-ci fait des allers-retours. Cet algorithme a une complexité de $O((m+n)n^2)$ étapes.

Malheureusement, cette approche requiert une connaissance totale du graphe et interdit ainsi son utilisation dans un contexte réparti.

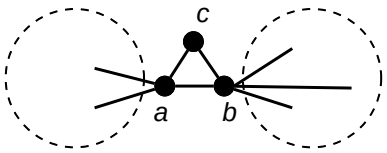


FIG. 3.11 – Illustration des faiblesses de la centralité d'intermédiarité : c obtient un score bas malgré son importance

3.2.4 Centralité du second ordre

Nous allons ici décrire l'algorithme réparti permettant d'obtenir la centralité du second ordre.

Intuition

Afin de comprendre le principe de notre algorithme, nous allons ici illustrer le principe de l'algorithme dans un contexte particulier et extrême, à savoir un graphe de type *barbell*. La figure 3.12 représente un tel graphe. Un graphe barbell est constitué de deux composantes complètement connectés de m_1 noeuds, connectés par un chemin de m_2 noeuds. Dans la figure 3.12, on a $m_1 = 5$ et $m_2 = 2$.

Considérons une marche aléatoire sur le graphe. Intéressons nous au noeud v_L de la figure 3.12 : lorsque la marche est dans la partie gauche du graphe, il a comme les autres noeuds une probabilité $1/m_1$ d'être la destination de la marche à chaque étape. Lorsque la marche aléatoire passe dans la partie de droite maintenant, v_L est le point de passage obligatoire, à l'aller comme au retour. Ainsi, les ponts v_L et v_R sont visités plus régulièrement par la marche. Cela se traduit par un écart type des temps de retour (c'est-à-dire le nombre de sauts entre deux visites sur le même

noeud) moindre. Nous avançons que les différences de rôles de ces noeuds se traduisent dans les écarts types des temps de passage.

De plus, il est facilement démontrable (voir p.ex. [AF]) qu'une marche aléatoire partant d'un noeud quelconque dans la partie gauche met en moyenne $m_1^2 m_2$ étapes à arriver à une autre noeud donné quelconque de la partie droite (et inversement) : la marche met du temps pour passer d'un côté à l'autre. Ainsi, lorsque la marche est dans une des parties, les noeuds de cette partie la voient passer régulièrement, et plus du tout lorsque cette marche change de côté. L'écart type des temps de passage sera fort, beaucoup plus fort que dans un graphe identique de $2m_1$ noeuds bien connectés. Ainsi, par l'observation (locale) de l'écart type des temps de passage de la marche, chaque noeud peut détecter la présence de chemins critiques et de pièges dans la topologie. Ceci clos l'illustration dans le cas du graphe barbell.

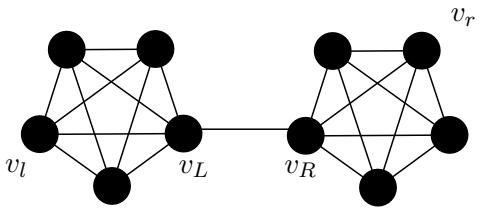


FIG. 3.12 – Exemple d'un graphe Barbell

L'algorithme

Avant de présenter l'algorithme, nous allons insister sur deux points clés de la conception de cet algorithme : le débiaisement de la marche et le calcul de l'écart type.

Marche aléatoire débiaisée Dans sa publication, Newman considère une marche aléatoire classique, dans le sens où la destination de la marche est prise uniformément au hasard dans les voisins du sommet courant. La *distribution stationnaire* (c'est-à-dire la probabilité de trouver la marche sur le sommet i après le temps de mixage, c'est à dire après un temps suffisamment long pour que cette probabilité ne dépende plus du point de départ de la marche) d'une telle marche est : $\pi_i = d_i/2m$ (voir *e.g.* [MR95]), c'est à dire que plus le noeud est connecté, plus il voit la marche. Newman a souligné ce défaut et a montré expérimentalement une corrélation entre le degré des noeuds et leur centralité d'intermédiarité basée sur les marches aléatoires, même pour les noeuds ayant une importance faible pour la topologie malgré un fort degré. On dit que la marche est *biaisée*.

Afin de pallier à ce problème et combattre l'effet d'une distribution des degrés très hétérogène, notre algorithme utilise une marche aléatoire *débiaisée*, dans le sens où la distribution stationnaire est indépendante du degré des noeuds : $\forall i \in V, \pi_i = 1/n$. Ainsi, après le temps de mixage, la marche a la même chance de se retrouver sur n'importe quel noeud. Autrement dit, après un nombre de pas suffisamment grand, tous les noeuds ont le même nombre de visites de la marche à exploiter.

Le fait d'apporter des temps de retour à tous les noeuds indépendamment de leur degré augmente la vitesse de l'algorithme. De plus, la marche débiaisée capture ainsi l'importance des noeuds indépendamment de leur degré. Pour débiaiser la marche, nous avons utilisé la technique dite de Metropolis-Hastings ([Has70, SRD+06]) : lorsque la marche est sur un noeud i, la destination est toujours choisie uniformément au hasard parmi γ_i, mais la marche n'est effectivement

transmise que selon une probabilité dépendant du degré de la destination et du degré de i. Ce processus est décrit dans les lignes 1 à 9 de la description de l'algorithme, figure 3.13.

Écart type des temps de retour Un des points clé de l'algorithme est le calcul des temps de l'écart type des temps de retour que nous détaillons ici. Chaque noeud i de $\mathcal{G}$, lors de la première réception de la marche aléatoire, crée un tableau Ξ_i, qui enregistre chaque temps de retour de la marche en i. Rappelons qu'un temps de retour est le temps mis par la marche aléatoire partant de i pour revenir en i. On note $\Xi_i(k)$ le k-ième temps de retour de la marche en i. Après le troisième passage de la marche, le noeud i calcule l'écart-type

$$\sigma_i(N) = \sqrt{\frac{1}{N}\Sigma_{k=1}^{N}\Xi_i(k)^2 - [\frac{1}{N}\Sigma_{k=1}^{N}\Xi_i(k)]^2}$$

des N valeurs de Ξ_i. Ces temps de retour étant des expériences indépendantes, on a grâce à la loi forte des grands nombres :

$$\lim_{N\longrightarrow\infty}\sigma_i(N) = \sigma_i.$$

Une fois que la marche a fait suffisamment de pas, les valeurs de σ représentent alors l'importance des noeuds dans le graphe : plus la valeur est faible plus le noeud est important.

Temps de convergence de l'algorithme Chaque noeud doit être visité un certain nombre de fois par la marche afin de calculer un écart type qui a du sens. Ainsi, le temps de convergence de l'algorithme est relié au *temps de couverture* du graphe $\mathcal{G}$. Ce temps de couverture est défini comme le nombre de pas nécessités par une marche aléatoire pour visiter chaque sommet de $\mathcal{G}$. Feige [Fei95a] montre que ce temps de couverture est compris entre $(1+o(1))n\ln n$ pour le graphe complet et $\frac{4}{27}n^3 + o(n^3)$ pour le graphe *lollipop* [Fei95b], c'est-à-dire un graphe complet de $n/2$ noeud lié à une ligne de $n/2$ noeuds. Le résultat pour le graphe complet tient dans le cas d'une marche débiaisée puisque tous les noeuds ont le même degré. Nous ne connaissons pas de borne haute dans le cas d'une marche débiaisée.

Le coût de notre algorithme est donc le coût de la couverture, multiplié par une constante afin que chaque noeud ait plusieurs temps de retour.

Les autres algorithmes basés sur les marches aléatoires (*e.g.* [Fei96] pour étudier la connectivité d'un graphe) nécessitent $O(n^3)$ étapes dans le pire cas (c'est-à-dire pour des graphes très dégénérés). En pratique, ce temps est bien moindre pour obtenir des résultats satisfaisants.

3.2.5 Modélisation des temps de retour

Cette partie présente une analyse théorique de l'algorithme réparti que nous venons de présenter. Il en résulte une formule générique pour calculer la centralité du second ordre (*i.e.* le vecteur σ) des noeuds d'un graphe d'entrée représenté par sa matrice d'adjacence.

Cette formule sera utilisée dans les simulation du protocole afin d'étudier la vitesse de convergence de celui-ci. De plus, cette formule est utile pour $1)$ un administrateur de système souhaitant étudier le comportement d'un système avant de le déployer $2)$ les noeuds du graphe pour anticiper le comportement de l'algorithme. Nous fournissons aussi à titre d'exemple l'application de cette formule à trois classes de graphes réguliers.

Analyse théorique

Nous cherchons ici une formule donnant l'écart type des temps de retour d'un noeud donné, à partir de la matrice de probabilité de transitions d'un graphe. Les temps de retour sont fonction de

1: **Upon reception of the Random Walk on node i:**
2: /* *Metropolis-Hastings random walk* */
3: Choose a neighbor j from Γ_i uniformly at random
4: Query j for d_j
5: Generate a random number $p \in [0, 1]$ uniformly
6: **if** $p \leq d_i/d_j$ **then**
7: forward the Random Walk to j
8: **else**
9: Random Walk remains at i
10: /* *Standard deviation* */
11: **if** first visit of the Random Walk on i **then**
12: Create array Ξ_i
13: **else**
14: Compute return time r since last visit
15: Add r to Ξ_i
16: **if** $|\Xi_i| \geq 3$ **then**
17: Compute standard deviation :
18: $\sigma_i(N) = \sqrt{\frac{1}{N}\Sigma_{k=1}^{N}\Xi_i(k)^2 - [\frac{1}{N}\Sigma_{k=1}^{N}\Xi_i(k)]^2}$

FIG. 3.13 – Calcul de la centralité du second ordre

la position du noeud dans le graphe. Cette idée est au coeur de notre approche : classiquement, les recherches dans le domaine procurent des bornes sur les temps de retour, et les petites variations que les noeuds peuvent ressentir fonction de leur position dans le graphe sont cachées par la notation en grand 0. Ces petites variations permettent de différencier les noeuds selon leur position dans le graphe.

Nous utilisons le modèle classique des chaînes de Markov à temps discret pour représenter la marche aléatoire sur le graphe d'entrée. Les états de la chaîne de Markov sont les noeuds (l'ensemble V). Nous présentons le cas général d'une marche classique en premier lieu, puis nous nous concentrons sur le cas de la marche débiaisée. Nous avons donc ici besoin de la matrice de probabilités de transition du graphe. Cette connaissance globale n'est pas nécessaire à l'algorithme réparti.

Soit $X = \{X_n,\ n \in \mathbb{N}\}$ une chaîne de Markov à temps discrets homogène et irréductible sur l'espace d'états fini S. On note $P = (P(i,j))_{i,j\in S}$ sa matrice de probabilité de transition. Pour chaque état $j \in S$, on note $\tau(j)$ le nombre de transitions (*i.e.* de pas de la marche) pour atteindre chaque état j, *i.e.*

$$\tau(j) = \inf\{n \geq 1 \mid X_n = j\}.$$

L'espace d'états S étant fini et X étant irréductible, X est récurrente, ce qui signifie que $\tau(j)$ est fini presque sûrement. On note $f_j^{(n)}(i)$ la distribution des $\tau(j)$ lorsque l'état initial de la chaîne X est i, c'est-à-dire, pour tout $n \geq 1$,

$$f_j^{(n)}(i) = \mathbb{P}\{\tau(j) = n \mid X_0 = i\}.$$

$f_i^{(n)}(i)$ représente ainsi la probabilité, partant de l'état i, que le premier retour à l'état i arrive après n transitions (donc la probabilité pour le noeud i d'avoir un temps de retour égal à n) et, pour $i \neq j$, $f_j^{(n)}(i)$ représente la probabilité, partant de i, que la première visite de l'état j se fasse à l'instant n. Ces probabilités sont données par le théorème de [Cin75] :

Théorème 5 *Pour chaque $i, j \in S$ et $n \geq 1$, on a*

$$f_j^{(n)}(i) = \begin{cases} P(i,j) & \textit{si} \quad n = 1 \\ \displaystyle\sum_{\ell \in S - \{j\}} P(i,\ell) f_j^{(n-1)}(\ell) & \textit{si} \quad n \geq 2. \end{cases} \tag{3.1}$$

Preuve

Par la définition de $f_j^{(n)}(i)$ on a, pour $n = 1$, $f_j^{(1)}(i) = P(i,j)$. Pour $n \geq 2$, on a

$$\begin{aligned} f_j^{(n)}(i) &= \mathbb{P}\{\tau(j) = n \mid X_0 = i\} \\ &= \mathbb{P}\{X_n = j, X_k \neq j, 1 \leq k \leq n-1 \mid X_0 = i\} \\ &= \sum_{\ell \in S - \{j\}} \mathbb{P}\{X_n = j, X_k \neq j, 2 \leq k \leq n-1, X_1 = \ell \mid X_0 = i\} \\ &= \sum_{\ell \in S - \{j\}} P(i,\ell) \mathbb{P}\{X_n = j, X_k \neq j, 2 \leq k \leq n-1 \mid X_1 = \ell\} \\ &= \sum_{\ell \in S - \{j\}} P(i,\ell) \mathbb{P}\{X_{n-1} = j, X_k \neq j, 1 \leq k \leq n-2 \mid X_0 = \ell\} \\ &= \sum_{\ell \in S - \{j\}} P(i,\ell) f_j^{(n-1)}(\ell), \end{aligned}$$

où la dernière et l'avant dernière égalités proviennent respectivement des propriétés de Markov et de l'homogénéité de la chaîne de Markov X. $\square_{\textit{Théorème}}$

Pour chaque état $j \in S$ et $n \geq 1$, on note $f_j^{(n)}$ le vecteur colonne contenant les valeurs $f_j^{(n)}(i)$ pour chaque $i \in S$. Pour chaque état $j \in S$, nous introduisons la matrice Q_j obtenue à partir de la matrice P en remplaçant la $j$ème colonne par des zéros :

$$Q_j(i,\ell) = \begin{cases} P(i,\ell) & \text{if} \quad \ell \neq j \\ 0 & \text{if} \quad \ell = j. \end{cases}$$

Nous introduisons aussi le vecteur colonne P_j contenant la $j$ème colonne de la matrice P, *i.e.* $P_j(i) = P(i,j)$. L'équation (3.1) peut alors s'écrire en notation matricielle sous la forme suivante :

$$f_j^{(n)} = \begin{cases} P_j & \text{si} \quad n = 1 \\ Q_j f_j^{(n-1)} & \text{si} \quad n \geq 2, \end{cases} \tag{3.2}$$

ce qui permet un calcul facile des vecteurs $f_j^{(n)}$. Nous définissons maintenant la matrice $M = (M(i,j))_{i,j \in S}$ par $M(i,j) = \mathbb{E}\{\tau(j) \mid X_0 = i\}$. $M(i,i)$ représente le temps moyen entre deux visites de X au sommet i, et pour $i \neq j$, $M(i,j)$ représente le temps moyen partant de l'état i, pour atteindre l'état j pour la première fois. La chaîne de Markov X étant irréductible, on a $M(i,j) < \infty$ pour tout couple $i, j \in S$ et

$$M(i,i) = \frac{1}{\pi_i},$$

où π_i est la $i$ème de la distribution π, qui est l'unique solution au système $\pi = \pi P$.

Pour calculer toutes les valeurs de M, nous introduisons le vecteur colonne M_j contenant la $j$ème colonne de la matrice M, *i.e.* $M_j(i) = M(i,j)$ et le vecteur colonne de un noté $\mathbb{1}$. Ces valeurs moyennes sont fournies par le résultat suivant :

Corollaire 1 *Pour chaque état $j \in S$, on a*

$$M_j = (I - Q_j)^{-1}\mathbb{1}. \tag{3.3}$$

Preuve En utilisant la relation (3.2), nous obtenons

$$\begin{aligned} M_j &= \sum_{n=1}^{\infty} n f_j^{(n)} \\ &= P_j + Q_j \sum_{n=2}^{\infty} n f_j^{(n-1)} \\ &= P_j + Q_j \left(\sum_{n=1}^{\infty} n f_j^{(n)} + \sum_{n=1}^{\infty} f_j^{(n)} \right) \\ &= P_j + Q_j \left(M_j + \mathbb{1} \right), \end{aligned}$$

et, puisque $P_j + Q_j\mathbb{1} = \mathbb{1}$, nous obtenons

$$M_j = Q_j M_j + \mathbb{1}.$$

La matrice Q_j est la sous-matrice de probabilités de transition d'une chaîne de Markov absorbante avec $|S|$ états transcients et un état absorbant, donc la matrice $I - Q_j$ est inversible. D'où

$$M_j = (I - Q_j)^{-1}\mathbb{1}.$$

□*Théorème*

En pratique, le vecteur colonne M_j est obtenu, pour chaque état $j \in S$ en résolvant le système linéaire $(I - Q_j)M_j = \mathbb{1}$.

Considérons maintenant le deuxième moment de $\tau(j)$. Soit la matrice $H = (H(i,j))_{i,j\in S}$ définie par $H(i,j) = \mathbb{E}\{\tau(j)^2 \mid X_0 = i\}$. $H(i,i)$ représente le deuxième moment du temps de retour de X à l'état i, et pour $i \neq j$, $H(i,j)$ représente le deuxième moment du temps de passage de i à j pour la première fois. Soit le vecteur colonne H_j contenant la $j$ème colonne de la matrice H, *i.e.* $H_j(i) = H(i,j)$. Ces valeurs sont obtenues par le résultat suivant :

Corollaire 2 *Pour chaque état $j \in S$, on a :*

$$H_j = (I - Q_j)^{-1}(I + Q_j)M_j.$$

Preuve A l'aide de la relation (3.2), on obtient

$$\begin{aligned} H_j &= \sum_{n=1}^{\infty} n^2 f_j^{(n)} \\ &= P_j + Q_j \sum_{n=2}^{\infty} n^2 f_j^{(n-1)} \\ &= P_j + Q_j \left(\sum_{n=1}^{\infty} n^2 f_j^{(n)} + 2\sum_{n=1}^{\infty} n f_j^{(n)} + \sum_{n=1}^{\infty} n f_j^{(n)} \right) \\ &= P_j + Q_j \left(H_j + 2M_j + \mathbb{1} \right) \\ &= Q_j H_j + 2Q_j M_j + \mathbb{1} \\ &= Q_j H_j + Q_j M_j + M_j, \end{aligned}$$

Or, d'après le corollaire 1, on a $Q_j M_j + \mathbb{1} = M_j$. D'où

$$H_j = (I - Q_j)^{-1}(I + Q_j)M_j.$$

□*Théorème*

En pratique, le vecteur colonne H_j est obtenu pour chaque état $j \in S$ en résolvant le système linéaire $(I - Q_j)H_j = (I + Q_j)M_j$.

L'écart type $\sigma(i)$ du temps de retour à l'état i dans la graphe cible est donc donné par la relation

$$\sigma(i) = \sqrt{H(i,i) - [M(i,i)]^2}. \tag{3.4}$$

Marches débiaisées Lorsque la marche est débiaisée, le graphe se comporte comme si tous les noeuds avaient le même degré d (informellement, Metropolis-Hastings ajoute des auto-boucles aux noeuds mal connectés pour augmenter leur degré). Ainsi $P(i,j) = 1/d$ si les noeuds i et j sont connecté dans le graphe, et 0 sinon. Cela implique que la matrice P est symétrique, donc bi-stochastique, *i.e.* $\mathbb{1}^t P = \mathbb{1}^t$, avec t l'opérateur de transposition. On a donc $\pi_i = 1/|S|$ et $M(i,i) = |S|$. Pour le deuxième moment $H(i,i)$ des temps de retour à l'état i, d'après le corollaire 2,

$$(I - Q_j)H_j = (I + Q_j)M_j.$$

En multipliant à gauche par 1^t, on obtient

$$\mathbb{1}^t H_j - \mathbb{1}^t Q_j H_j = \mathbb{1}^t M_j + \mathbb{1}^t Q_j M_j. \tag{3.5}$$

Par définition de Q_j, on a $\mathbb{1}^t Q_j = \mathbb{1}^t - e_j$, où e_j est le $j$ème vecteur unité en ligne, *i.e.* $e_j(i) = 1$ si $i = j$ et 0 sinon. Donc l'équation (3.5) se simplifie en :

$$\mathbb{1}^t H_j - (\mathbb{1}^t - e_j)H_j = \mathbb{1}^t M_j + (\mathbb{1}^t - e_j)M_j$$

et donc

$$H(j,j) = 2\sum_{i \in S} M(i,j) - M(j,j) = 2\sum_{i \in S} M(i,j) - |S|.$$

L'écart type $\sigma(j)$ s'écrit donc, à partir de la relation (3.4),

$$\sigma(j) = \sqrt{2\sum_{i \in S} M(i,j) - |S|(|S|+1)}. \tag{3.6}$$

Résultat pour 3 classes de graphes

Nous instancions les formules des paragraphes précédents sur trois classes de graphes aux diamètres extrêmes : un graphe complet (diamètre 1), un anneau (diamètre $\lfloor \frac{n}{2} \rfloor$) et une ligne (diamètre n). Sur tous les graphes, on considère le cas de la marche débiaisée.

Pour l'anneau, on a $d = 2$ et les probabilités de transition non nulles sont données, pour tout état $i \in S = \{0, \ldots, n-1\}$, par

$$P(i, i+1 \ (mod\ n)) = P(i, i-1 \ (mod\ n)) = 1/2.$$

L'écart type des temps de retour est donné par le théorème suivant :

Théorème 6 *Pour une marche débiaisée sur un anneau de n noeuds, on a $\sigma(j) = \sigma$ pour tout j, avec*

$$\sigma = \sqrt{\frac{n(n-1)(n-2)}{3}}.$$

Preuve Il est facile de vérifier que la solution à l'équation (3.3) est donnée, pour $i \neq j$, par

$$M(i,j) = (n - |i-j|)(|i-j|)$$

et, vu que $M(i,i) = n$, on a

$$\sum_{i=0}^{n-1} M(i,j) = n + \frac{(n-1)n(n+1)}{6}$$

En utilisant l'équation (3.6), on obtient le résultat désiré. $\square_{Théorème}$

Le fait que tous les $\sigma(j)$ soient égaux est dû à la régularité de la structure. C'est une bonne nouvelle : tous les noeuds ont topologiquement parlant le même rôle, ils ont aussi la même importance.

Sur le graphe complet, on a $d = n-1$ et ainsi, pour tout couple d'états $i, j \in S$ la probabilité de transition est : $P(i,j) = 1/(n-1)$ si $i \neq j$ et $P(i,i) = 0$. Le théorème suivant donne l'écart type des temps de retour :

Théorème 7 *Pour une marche aléatoire débiaisée sur un graphe complet de n noeuds, on a $\sigma(j) = \sigma$ pour tout j avec*

$$\sigma = \sqrt{(n-1)(n-2)}.$$

Preuve Il est facile de vérifier que la solution à l'équation (3.3) est donnée, pour $i \neq j$, par

$$M(i,j) = n - 1$$

et, vu que $M(i,i) = n$, on a

$$\sum_{i=0}^{n-1} M(i,j) = n + (n-1)^2$$

En utilisant l'équation (3.6), on obtient le résultat désiré. $\square_{Théorème}$

A nouveau, le fait que tous les $\sigma(j)$ soient égaux est dû à la régularité de la structure.

Pour la ligne, on a $d = 2$ et les probabilités de transition non nulles sont données, pour tout état $i \in \{1, \ldots, n-2\}$, par

$$P(i, i+1) = P(i, i-1) = 1/2$$

et $P(0,0) = P(0,1) = P(n-1, n-2) = P(n-1, n-1) = 1/2$. Le théorème suivant donne le l'écart type des temps de retour :

Théorème 8 *Pour une marche aléatoire débiaisée sur une ligne de n noeuds, on a pour tout état $j \in \{0, 1, \ldots, n-1\}$,*

$$\sigma(j) = \sqrt{\frac{n(n-1)(4n-5)}{3} - 4nj(n-j-1)}.$$

Preuve

Il est facile de vérifier que la solution à l'équation (3.3) est donnée, pour $i \neq j$, par

$$\begin{aligned} M(i,j) &= (j-i)(i+j+1) \text{ pour } i < j \\ M(i,j) &= (i-j)(2n-(i+j+1)) \text{ pour } i > j \end{aligned}$$

et, vu que $M(i,i) = n$, on a

$$\sum_{i=0}^{n-1} M(i,j) = \frac{n(2n^2 - 3n + 4)}{3} - 2nj(n - j - 1).$$

En utilisant l'équation (3.6), on obtient le résultat désiré. □*Théorème*

Il est intéressant d'observer que grâce à la symétrie de la structure, on a $\sigma(j) = \sigma(n-j-1)$. De plus $\sigma(j)$ est minimal pour les noeuds $\lfloor \frac{n-1}{2} \rfloor$ et $\lceil \frac{n-1}{2} \rceil$. Cette remarque souligne le fait que notre algorithme produit effectivement un résultat de type « centralité » puisque les noeuds critiques d'une ligne (par rapport aux centralités) sont les noeuds du milieu. C'est aussi ces noeuds du milieux qui verraient leurs liens rompus dans le cas d'un calcul de conductance.

Dans le contexte des graphes réguliers, l'anneau et le graphe complet sont deux cas extrêmes de robustesse de structure : l'anneau est la plus faible (il est seulement 2-connexe) et le graphe complet est la structure la plus solide (il est $(n-1)$-connexe). Nous pensons qu'il est raisonnable de supposer que l'écart type des graphes évolue entre ces bornes : entre $o(n)$ pour le graphe complet et $o(n^{3/2})$ pour l'anneau.

Application à des graphes spécifiques : signatures

Nous avons vu que l'observation des centralités permet de distinguer, dans un graphe donné, les noeuds importants des noeuds périphériques. Nous allons ici voir une autre application des centralités : les signatures de graphes. Nous définissons la signature d'un graphe comme la distribution des centralités du second ordre des noeuds le composant. A l'aide de la formule (3.4) nous allons ici présenter les signatures de différents graphes types que l'on peut rencontrer dans les systèmes répartis. Nous verrons comment les propriétés théoriques de ces graphes se retrouvent dans l'observation des signatures.

Les valeurs de σ sont calculées pour chaque noeud, à partir de la matrice de probabilités de transition, sur la base d'une marche aléatoire débiaisée.

En entrée, nous considérons les graphes suivants :

Aléatoire il s'agit d'un graphe généré selon le modèle Erdös-Rényi [ER59], où chaque paire de noeuds est connectée avec une probabilité $p = \frac{\theta \ln n}{n}$. Avec le paramètre $\theta > 1$, le graphe est connexe.

Clusterisé ce graphe est construit à partir de deux graphes aléatoires de même taille reliés par un seul pont. Le graphe résultant ressemble au graphe Barbell présenté plus haut, mis à part que les deux clusters n'ont pas la connectivité complète.

Ring-Lattice ce graphe est un graphe circulaire où un noeud i est connecté aux noeuds $i-k/2, i-k/2+1, \ldots, i+k/2$. Les valeurs sont définies modulo n, et k est un paramètre du graphe.

Scale-free il s'agit d'un graphe généré selon le modèle Barabási/Albert [AB02]. On parle aussi d'attachement préférentiel. Ce graphe est utilisé pour décrire de nombreux réseaux sociaux.

Nous avons sélectionné ces graphes car nous pensons qu'il représentent des familles classiques de graphes, c'est-à-dire largement étudiés et/ou souvent utilisés dans les réseaux d'aujourd'hui ou dans l'étude des graphes sociaux.

Tous les graphes étudiés comportent 10^3 noeuds et ont été générés avec des paramètres assurant un degré de noeud moyen de 20 voisins, afin d'obtenir une comparaison équitable.

La figure 3.14 présente les résultats sous forme d'histogrammes. Un point particulier, mettons $(x = 2500, y = 3)$, traduit le fait que trois noeuds du graphe ont une centralité du second ordre (*i.e.* un σ) de 2500.

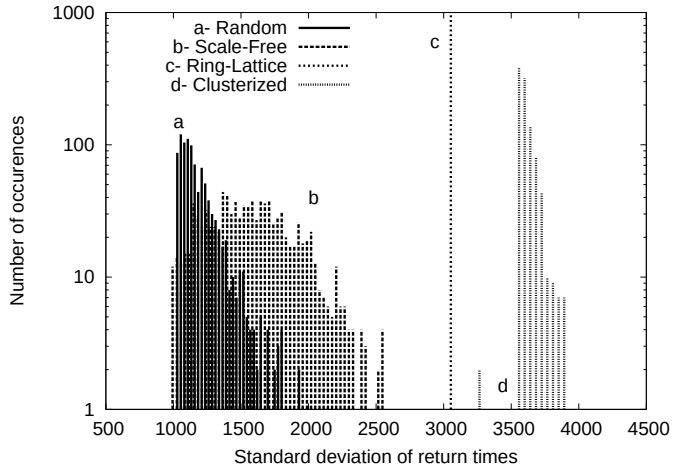


FIG. 3.14 – Histogramme de la valeur théorique de la centralité du second ordre pour 4 graphes particuliers

Ce qui frappe en premier est la disparité des distributions. Une distribution étroite signifie que tous les noeuds ont un rôle semblable dans la structure. Au contraire, des valeurs éparpillées traduisent une forte disparité des rôles dans la structure. De même, un σ moyen élevé traduit un graphe dont la navigabilité est médiocre (à cause d'un diamètre grand ou de la présence de goulots d'étranglement).

Le graphe ayant les plus basses valeurs de σ est le graphe aléatoire (noté *a*), avec des valeurs majoritairement comprises dans la fourchette $[1000 : 1500]$. Ceci correspond au consensus de la communauté sur les bonnes propriétés de cette classe de graphes : petit diamètre et petit coefficient de clusterisation. Ensuite vient le graphe Barabási/Albert (*b*), dont les valeurs s'étendent sur une plage plus large ($[1000 - 2500]$). La aussi, il est reconnu que ces graphes ont une structure présentant une forte disparité des rôles : les noeuds de fort degré (aussi appelés *hubs*) font partie de nombreux plus courts chemins et les autres noeuds ont une importance bien moindre. Pour le ring-lattice (*c*), la distribution est une ligne (d'où l'utilisation d'une échelle logarithmique pour les ordonnées) qui traduit la parfaite régularité de la structure. Cette ligne se situe au dessus de 3000, ce qui s'explique par le diamètre élevé (50) de la structure. Enfin, le graphe clusterisé (*d*) présente une distribution intéressante pour deux raisons. La première est que cette structure pourtant composée de deux graphes aléatoires exhibe une valeur de σ moyenne trois fois plus importante que le simple graphe aléatoire. Ceci est bien sûr expliqué par la difficile navigation de la marche aléatoire dans le graphe, comme expliqué dans la section 3.2.4. La deuxième porte sur la ligne détachée autour de 3250 : les deux noeuds possédant cette valeur sont les ponts (comme les noeuds v_L et v_R de la figure 3.12). Cela confirme l'intuition de la section 3.2.4 : ces noeuds jouent un rôle important dans la topologie et sont donc visités plus régulièrement par la marche aléatoire.

La centralité de second ordre se montre ainsi un outil utile pour la caractérisation des graphes et permet de les classer en fonction de leur robustesse et de leur navigabilité.

3.2.6 Éxpérimentation

Dans cette partie, nous présentons l'évaluation de l'algorithme réparti présenté en 3.2.4. Nous avons évalué notre algorithme à travers plusieurs métriques.

1. Sa capacité à produire des résultats qui ne sont pas biaisés par des distributions de degrés hétérogènes.

2. La correspondance entre l'étude théorique et les résultats produits par l'algorithme. Ceci nous permettra de plus d'observer la vitesse de convergence de l'algorithme vers la valeur théorique. Nous observerons que nous sommes bien loin des bornes hautes fournies par la théorie.
3. La capacité de l'algorithme à détecter des noeuds critiques dans une topologie donnée.

En utilisant les résultats de l'analyse théorique, nous montrons ainsi que l'algorithme converge vers la prédiction théorique, à une vitesse raisonnable. Nous illustrons ensuite les utilisations possibles des valeurs ainsi obtenues pour la maintenance du réseau ou la découverte de zones clusterisées.

Les expériementations ont été réalisées à l'aide du simulateur d'évènements discrets Peersim [pee].

Degré des noeuds et débiaisement

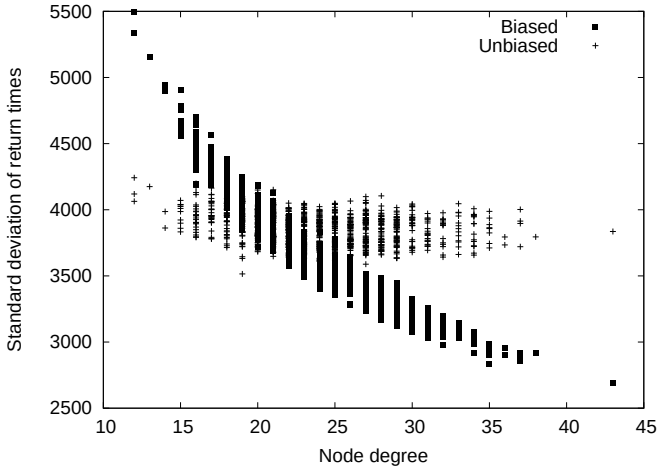


FIG. 3.15 – Centralité du second ordre : marche débiaisée contre marche aléatoire classique

Nous montrons ici les effets du débiaisement sur les centralités du second ordre obtenues. Pour cela, nous avons calculé les centralités du second ordre de manière débiaisée (c'est-à-dire selon le code fourni figure 3.13) et de manière classique (c'est-à-dire avec une marche aléatoire classique, non débiaisée par le méthode de Metropolis-Hastings), dans le même graphe. Ce graphe est un graphe clusterisé de 1000 noeuds identique à celui présenté en section 3.2.5, avec $p = \frac{1.5 \ln n}{n}$. La figure 3.15 représente les valeurs des centralités des noeuds après 2.10^6 pas.

On peut observer dans le cas de la marche non débiaisée une corrélation forte entre le degré et les valeur de centralité. Rappelons qu'un σ faible signifie une importance forte. Dans le cas non débiaisé, tous les noeuds de fort degré ont une valeur de σ faible, indépendamment de leur importance réelle. A l'opposée, dans le cas débiaisé les valeurs de σ sont concentrées dans une plage plus réduite et la distribution des valeurs n'exhibe pas de corrélation avec le degré. Ceci prouve l'effet du débiaisement : il évite de considérer les noeuds de plus haut degré comme plus importants qu'ils ne le sont réellement.

Notons que le débiaisement préserve tout de même une légère corrélation entre le degré des noeuds et leur importance : statistiquement plus un noeud a de liens, plus il a de chances d'avoir une fonction importante (par exemple en jouant un rôle de hub et se retrouvant ainsi sur beaucoup de plus courts chemins [New05]).

Vitesse de convergence

Cette section étudie la vitesse de convergence du protocole vers les valeurs théoriques calculées avec la formule (3.4).

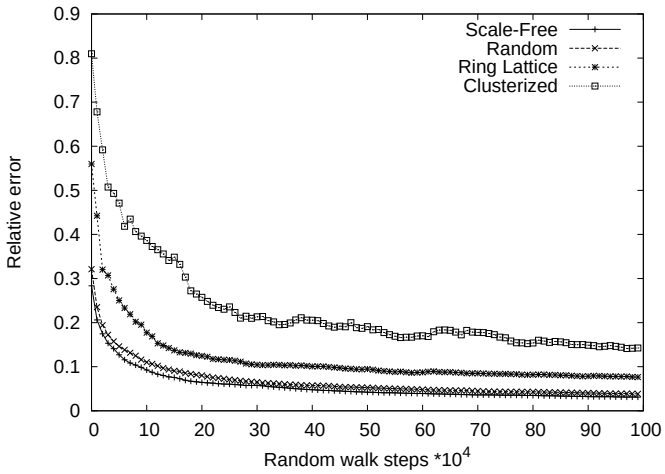


FIG. 3.16 – L'algorithme converge vers les valeurs théoriques, pour 4 classes de graphes

La figure 3.16 représente le ratio d'erreur de l'algorithme à mesure que l'algorithme progresse. En abscisses nous avons le nombre de pas de la marche aléatoire et en ordonnés, l'erreur relative, c'est-à-dire en reprenant les notations de la section 3.2.4

$$\sum_{i \in V} \frac{|\sigma_i(N_i) - \sigma_i|}{\sigma_i}$$

où N_i représente le nombre de temps de retours collectés par le noeud i. Les courbes présentées sont la moyenne de 20 expériences indépendantes.

A mesure que l'algorithme progresse, le nombre de valeurs collectées par chaque noeud augmente, améliorant la précision de l'estimation. On observe que l'algorithme converge rapidement avec une petite marge d'erreur. Ceci valide l'étude théorique, et on observe une convergence rapide comparée au pire cas $O(n^3)$ du temps de couverture. Il est intéressant d'observer que moins le graphe est navigable, plus la convergence est lente : dans le cas de graphes à haut diamètre, ou clusterisés, la marche aléatoire peut rester bloquée dans certaines zones. Elle met du temps à sortir de ces « pièges » et cela impacte le temps de convergence de l'algorithme.

Néanmoins, si l'on considère ces résultats conjointement avec ceux de la figure 3.14, on déduit que les noeuds peuvent rapidement détecter le type de graphe dans lequel ils se trouvent. En effet, rappelons que le σ moyen d'un noeud dans un graphe aléatoire et trois fois moins grand que celui d'un noeud dans un graphe clusterisé. Ainsi, même avec une erreur de 10%, la différence entre les σ est telle que les noeuds peuvent conclure avec assurance s'ils se trouvent dans un graphe en bonne santé ou pas.

Identification des goulots d'étranglement

Nous présentons maintenant une application particulière de l'algorithme : l'identification de noeuds critiques dans la topologie. A cette fin, nous nous intéressons à la détection des ponts. L'identification de ces ponts est très importante puisque ces noeuds, à cause de leur importance stratégique, sont très sollicités et sont les dernières chances d'éviter le partitionnement du graphe

qui est un évènement rarement désirable dans un réseau. Les identifier permet de réagir en re-maillant la zone du graphe autour du noeud identifié par exemple.

Pour cette expérience, nous utilisons le graphe clusterisé présenté dans les sections 3.2.6 et 3.2.5, formé de 2 clusters de 500 noeuds reliés par un pont. L'expérience consiste à regarder l'évolution de la centralité des noeuds assurant le pont par rapport aux centralités du reste des noeuds. Ces noeuds étant importants, leurs centralités doivent être les plus basses (comme le montre la théorie sur la figure 3.14).

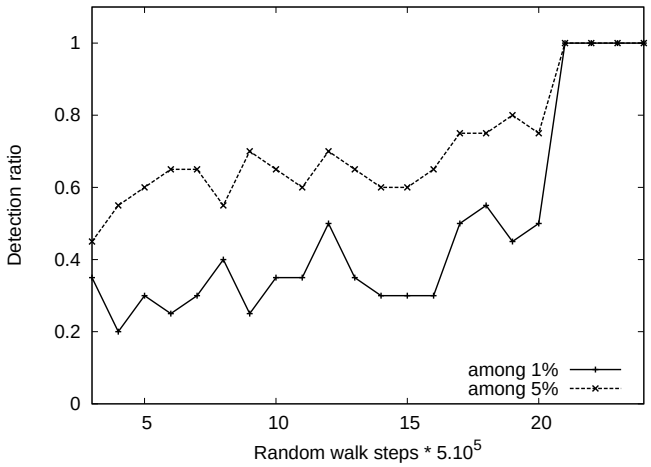


FIG. 3.17 – Evolution de la position des ponts parmi les plus basses centralités du réseau

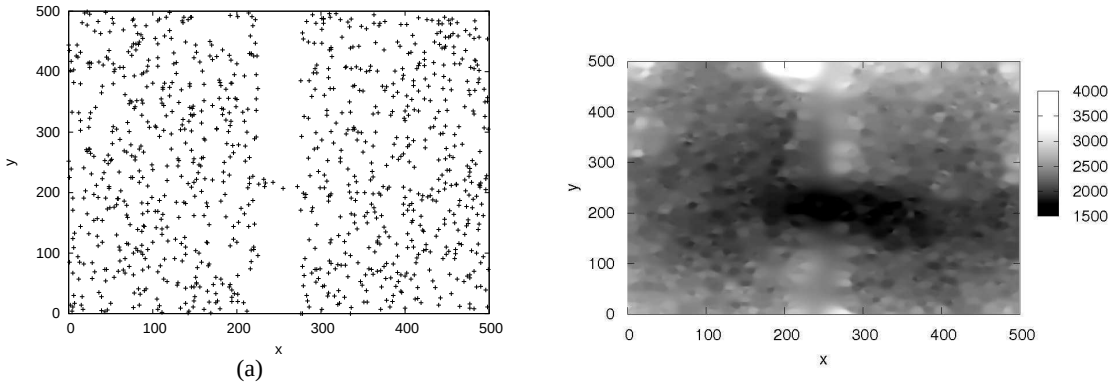
La figure 3.17 décrit la position effective de ces noeuds dans les 1% et 5% des centralités les plus basses du réseau en fonction du nombre de pas de la marche aléatoire. Chaque point est le résultat de 20 expériences indépendantes. Ainsi, le point $(x = 5.10^6, y = 0.35)$ sur la courbe des 1% signifie qu'après 5.10^6 pas, un des deux ponts se retrouvait dans les 1% des plus basses valeurs du réseau dans 35% des cas.

La bonne nouvelle est la convergence de l'algorithme vers la détection certaine de ces ponts, conformément à l'étude théorique. La moins bonne est le temps de convergence relativement long. Ceci s'explique à nouveau par l'extrême difficulté qu'éprouve la marche aléatoire à aller d'un cluster à l'autre.

Les figures 3.18(a) et 3.18(b) illustrent visuellement cet effet. La figure 3.18(a) représente la disposition d'un réseau en deux dimensions, à la réseau de capteurs : chaque point représente un noeud, et chaque noeud est connecté aux noeuds géographiquement proches (ses voisins sont les noeuds présents dans un disque de 50 unités de distance). Les résultat de l'exécution de notre algorithme, après 10^6 pas, sont représentés sur la figure 3.18(b) : les zones sombres sont des zones où la centralité du second ordre est basse, les zones claires sont celles ou la centralité est forte. On observe clairement une corrélation entre la position de noeuds (notamment leur distance au pont) et la valeur de leur centralité. A l'opposée, les noeuds situés dans les coins obtiennent des valeurs de centralité très forte, reflétant leur faible importance. A l'adresse [vid] figure une animation de l'évolution des valeurs de σ des noeuds, avec des images prises tous les 25.10^3 pas.

Vers une détection répartie

Nous explorons ici comment exploiter les informations récoltées de manière répartie afin d'éliminer toute implication d'un acteur central.



(a) (b)

FIG. 3.18 – (a) Répartition de 10^3 noeuds sur une topologie 2-D comportant un goulot d'étranglement, (b) Distribution des écarts types, après 1×10^6 pas

Il est d'abord possible de fournir aux noeuds un σ_{ideal} ainsi qu'une fourchette d'erreur, pour permettre aux noeuds de décider eux-mêmes de lancer des réparations s'ils se sentent dévier de la valeur cible. Calculer cette valeur σ_{ideal} nécessite de connaître quelques paramètres du système ainsi asservi, tel le nombre de noeuds du système et le degré moyen ciblé. Ensuite, le calcul de σ_{ideal} peut s'effectuer à l'aide des modèles mathématiques de la section 3.2.5, ou à l'aide de simulations.

Les noeuds peuvent aussi détecter la présence de clusters en échangeant leurs tableaux de temps de passages Ξ. Supposons que deux noeuds a et b échangent leurs ensembles Ξ_a et Ξ_b. Il est possible d'utiliser le ratio $r_{a\to b} = \frac{\sigma(\Xi_a \cup \Xi_b)}{\sigma(\Xi_a)}$ pour détecter de manière répartie la présence de clusters : si a et b sont dans deux clusters différents, alors l'écart type de l'union de leur temps de passage sera petite par rapport à leur σ. Le ratio $r_{a\to b}$ est bas. Inversement, si a et b sont situés dans le même cluster, le ratio $r_{a\to b}$ sera proche de 1 puisque la marche aléatoire visitera les deux noeuds dans les mêmes périodes.

Pour illustrer ce principe, considérons un graphe constitué de 2 clusters A et B. Une marche aléatoire démarrant dans A reste dans le cluster pendant 2000 sauts, puis passe dans B pour encore 2000 sauts, avant de revenir dans A pour 2000 sauts, ainsi de suite. Un noeud a du cluster A verra la marche régulièrement dans les périodes $[0 : 2000]$ et $[4000 : 6000]$. Ainsi ce qui rend $\sigma(\Xi_a)$ grand, c'est la grosse période d'absence de la marche entre la dernière visite du premier intervalle et la première visite du deuxième. Si a réunit sont tableau avec celui d'un noeud b situé dans le cluster B, et qui a donc vu la marche régulièrement dans l'intervalle $[2000 : 4000]$, cette grosse période d'absence disparait, donc $\sigma(\Xi_a \cup \Xi_b)$ est petit par rapport à $\sigma(\Xi_a)$. Inversement, si b est dans le cluster A aussi, la période d'absence de la marche sera la même pour les deux noeuds, l'union des tableaux n'améliorera rien.

L'opération d'union des deux tableaux consiste à virtuellement fusionner les noeuds : s'ils appartiennent au même cluster, peu de changements ; s'ils appartiennent à des clusters différents, le « noeud virtuel » devient un pont avec une centralité faible.

La figure 3.19 illustre ce phénomène. A partir de l'exécution de l'algorithme sur un graphe de 1000 noeuds, la figure 3.19 représente les valeurs des ratios $r_{a\to b}$ pour tous les couples de noeuds. Dans ce graphe, les 500 premiers noeuds appartiennent au premier cluster, et les 500 derniers au deuxième cluster. Les deux clusters sont reliés par un pont entre les noeuds d'identité 499 et 500. Sur la figure, la couleur du point $(x = 200, y = 800)$ représente la valeur du ratio $r_{200\to 800}$. Une couleur sombre représente un ratio faible et une couleur claire représente un ratio faible. La

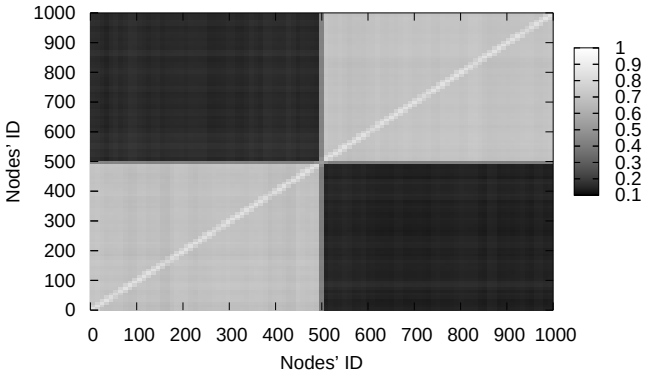


FIG. 3.19 – Comparaison noeud à noeud des temps de passage pour un graphe clusterisé

symétrie de la figure est due à la symétrie de l'opération d'union : on a $r_{a\to b} \equiv r_{b\to a}$. On observe que les noeuds du même cluster ont des ratios élevés et les noeuds de clusters différents ont des ratios faibles.

Ainsi, à l'aide d'un mécanisme de sampling (*e.g.* [VGS05, JGKvS04]), chaque noeud peut détecter si un autre noeud se situe dans son cluster ou pas. Ceci constitue un renseignement précieux pour d'éventuelles opérations de réparation du graphe.

3.3 Conclusion

Nous avons vu deux études au cours de ce chapitre. Ces deux études ont été publiées dans leurs versions originales : [MRT09] et [KMST08] (voir la version complète [KLMST09]).

La première étude a pour objectif de mettre en lumière l'impact des couches sous-jacentes dans les performances d'une application. Cette partie illustre aussi, grâce au protocole proposé, que la compréhension des phénomènes dus à la couche inférieure permet de minimiser leur impact efficacement. Les contributions de cette partie sont les suivantes :

- Nous proposons un nouveau modèle de défaillances. Ce modèle ne repose pas sur un nombre maximal de processus pouvant crasher, mais propose de modéliser plus finement les crashs dans le temps à l'aide de la fonction que nous nommons $\alpha()$.
- Nous proposons un algorithme réparti estimant dynamiquement le nombre de processus vivants d'un système à passage de messages basé sur le modèle de crashs $\alpha()$. Ce protocole, simple, termine toujours et ne nécessite pas la connaissance du nombre d'entités présentes dans le système.
- Nous simulons ce protocole sur différentes topologies. Ces simulations montrent que l'estimation réalisée par le protocole est de bonne qualité.

  Ces simulations permettent en outre d'observer le partitionnement de certains réseaux très hétérogènes. En effet les configurations mettant en défaut le protocole sont des réseau dont la structure de communication est elle-même déjà très clusterisée : plusieurs grappes regroupant des processus communiquant très bien entre eux, reliées par des liens lents. Dans ces types de structures, les échanges d'information locaux, intra-cluster, sont tellement prédominants par rapport aux échanges d'information inter-clusters, que les processus d'un cluster donné peuvent suspecter tous les processus extérieurs à ce cluster.

– Enfin, pour répondre à ce constat, nous proposons un nouveau protocole qui améliore sensiblement les performances (en vitesse de convergence) et la robustesse de l'évaluation du nombre de processus vivants dans les réseaux hétérogènes.

C'est en utilisant l'impact la structure de communication sur une application dédiée (une marche aléatoire permanente) qu'au cours de la deuxième partie nous illustrons comment étudier et donner conscience aux noeuds de l'état de la couche sous-jacente qu'ils utilisent. Les contributions de cette partie sont les suivantes :

– Nous proposons une nouvelle centralité, c'est-à-dire une nouvelle mesure de l'importance d'un noeud dans une topologie donnée. Cette centralité, la centralité du second ordre, repose sur l'observation des temps de retour d'une marche aléatoire et ainsi ne souffre pas des inconvénients des autres centralités.
  A l'aide d'une analyse, nous montrons que cette centralité a du sens (*i.e.* elle permet de détecter les noeuds importants). Dans [LMT09], nous comparons les différents types de centralités sur un réseau social type. Les résultats montrent que notre centralité se comporte bien, voire mieux que la majorité des centralités comparées.
– Nous proposons un protocole entièrement réparti permettant à chaque noeud d'un grand réseau d'estimer sa centralité du second ordre. Ce protocole repose sur un mécanisme très simple : une marche aléatoire parcourt perpétuellement le réseau. Chaque noeud, par l'observation des temps de passage de cette marche, peut estimer son importance. En particulier, ce protocole ne nécessite aucune centralisation ni horloge synchronisée.
– A l'aide de simulations, nous montrons que le protocole permet efficacement de calculer la centralité du second ordre. Nous montrons aussi qu'il est possible d'utiliser cet algorithme pour établir la signature d'un graphe et ainsi identifier le type de graphe que constitue la structure de communication sous-jacente.
  Nous montrons comment il est possible d'employer ce protocole pour identifier des points chauds de la topologie (*i.e.* les ponts entre deux clusters). Nous montrons qu'il est aussi possible pour un noeud, en exploitant les temps de passage de la marche aléatoire, de savoir si un autre noeud donné au hasard appartient au même cluster que lui ou pas. Cette indication est très utile pour la réparation d'une topologie dégénérée.

Enfin, ajoutons que ces deux protocoles peuvent être complémentaires. Rappelons que la fonction $\alpha_i()$ du premier protocole représente le maximum d'entités pouvant être déconnectées de i par unité de temps. Or la centralité procurée aux noeuds par le deuxième protocole représente la santé de la topologie, ou encore la robustesse de la strucutre de communication. Cette robustesse a un impact direct sur le nombre d'entités pouvant être déconnectées du noeud i. Autrement dit, il est possible d'alimenter la fonction $\alpha_i()$ du premier protocole à l'aide de la centralité σ_i du noeud i procurée par le second protocole.

Chapitre 4

Exploiter les structures

Dans le chapitre 3, l'objet de notre étude était la structure du système : nous avons vu que la structure d'un système influait de manière considérable sur les performances d'une application rudimentaire exploitant ce système. Partant de ce constat, nous avons étudié les manières de capturer cette influence et de la mesurer grâce à la centralité.

Dans cette partie, nous allons changer le regard porté sur les structures des systèmes répartis. Il s'agit maintenant de s'en servir en tant qu'outil pour réaliser l'application désirée dans le système. En effet, la structure est par définition un ensemble de contraintes liant les entités. Nous allons voir comment reposer sur ces contraintes pour construire des solutions.

Pour continuer notre tour d'horizon des systèmes répartis, ce chapitre donnera la part belle à une adversité évoquée en introduction : les byzantins. Les byzantins sont un terme générique pour évoquer la présence d'un adversaire dans le système, dont le but est faire faillir l'application, ou faire dévier le système du comportement souhaité par l'administrateur. A l'origine, il s'agit d'une image employée par Leslie Lamport [LSP82] pour introduire un modèle où le crash d'une entité ne se traduit pas par l'arrêt de celle-ci mais par un comportement arbitraire. Afin de résister à n'importe quel comportement, il faut résister au pire des comportements, c'est pourquoi lors de la mise au point de protocoles résistants à de tels crashs il est commode d'imaginer un adversaire puissant et bien informé.

Par la suite, avec le développement de réseaux plus ouverts (réseaux pair-à-pair, réseaux de capteurs), de nouveaux types de comportements déviants sont apparus, ce qui a mené à différentes définitions de l'adversité pouvant apparaître dans ces réseaux. Dans cette section, nous allons ainsi voir deux types d'adversité :

- Dans la première étude, nous supposons un adversaire byzantin classique : cet adversaire possède une connaissance illimitée, une capacité de calcul presque illimitée. Cet adversaire possède t entités qu'il peut contrôler à sa guise, avec pour objectif d'empêcher les $n - t$ unités correctes du système de se mettre d'accord (c'est-à-dire exécuter un consensus).
- La deuxième étude porte sur les réseaux de capteurs. A ce titre, les entités composant les réseaux sont faibles et limités : elles ont peu de capacité de calcul, peu de mémoire et peu d'énergie pour calculer et émettre des messages. Dans ce contexte, le modèle classique de l'adversaire byzantin est beaucoup trop fort. Nous utilisons donc un modèle d'adversité affaibli : nous supposons par exemple qu'il n'existe pas un unique adversaire contrôlant toutes les entités, mais plutôt que chaque entité est un adversaire. Nous supposons de plus que ces entités ennemies ne collaborent pas. Dans cette étude, nous tentons de permettre aux entités de forger des clés afin de communiquer de manière sécurisée, malgré des communications facilement interceptables.

Pour parvenir à nos fins, nous allons dans les deux cas utiliser les structures proposées par les modèles du système. Ici à nouveau, les entités sont discriminées selon leur capacité à com-

muniquer, ce qui engendre une structure (la structure formée par les relations de communication privilégiées). Les motifs de discrimination sont différents dans les deux études :

- Dans la première étude, nous sommes dans un système partiellement asynchrone. Ainsi, chaque processus peut envoyer un message à tous les autres, mais ce message n'arrivera pas forcément à temps à destination ; le cas échéant il sera ignoré par le destinataire. Afin de parvenir au consensus (un problème fondamental d'accord qui sera défini formellement par la suite), la propriété que le système doit vérifier est qu'un processus au moins aura au bout d'un certain temps des communications synchrones avec une partie du reste du système. C'est donc à ce titre une contrainte sur la structure des échanges d'informations que nous utilisons pour bâtir notre solution.
- Dans la deuxième étude, le système est synchrone et les processus sont discriminés par leurs interlocuteurs : chaque processus ne peut communiquer qu'avec une petite sous-partie du système. Nous allons utiliser les particularités du graphe de communications pour permettre à certains noeuds du système d'échanger des données de manière sécurisée. Nous verrons que dans ce cas, les propriétés que la structure doit garantir sont locales : la communication sécurisée est possible dans tous les endroits où la structure a localement les bonnes propriétés.

4.1 Consensus byzantin dans un réseau très asynchrone

Dans cette section nous allons présenter et prouver un protocole réalisant le consensus dans un système asynchrone composé de n processus. Parmi ces n processus, t peuvent exhiber un comportement byzantin, c'est-à-dire qu'ils peuvent se comporter arbitrairement.

4.1.1 Motivation

Dans un système réparti, un processus est dit correct si son comportement suit sa spécification durant toute l'exécution. Un processus peut aussi dévier de sa spécification de différentes manières. Cette dérive peut être un simple arrêt, c'est-à-dire qu'à partir d'un certain point le processus stoppe toute exécution et ne fait plus rien, il se bloque. On parle alors de modèle de défaillance crash-stop. Un processus peut dériver de sa spécification en exhibant un comportement qui n'est pas celui demandé. On parle alors de comportement *byzantin*. Ce comportement peut être causé par de multiple raisons. Les plus fréquemment évoquées sont :

La capture Un des processus a été piraté par des personnes mal intentionnées. Ces personnes ont changé le code exécuté par le processus par un code de leur choix. L'inconvénient ici est que l'adversaire a accès aux données déjà contenues dans l'entité (telles des clés cryptographiques par exemple).

Les erreurs humaines Le champ est large : d'une erreur de programmation à une erreur de configuration ou de déploiement [Joh03].

Les erreur « transitoires » Il s'agit ici d'erreurs hardware provoquées par exemple par des sources de rayonnement, ou encore par des interférences électriques. Ces erreurs peuvent corrompre par exemple un bit de mémoire et amener le processus a un comportement complètement différent du comportement prévu initialement.

Il est important d'observer que le modèle de défaillances byzantin est beaucoup plus général que le modèle de défaillance classique (un byzantin peut, entre autres, se comporter comme un processus crashé du modèle crash-stop, alors que le contraire est faux). Ainsi le modèle byzantin signifie que les processus peuvent crasher de n'importe quelle manière.

Nous nous intéressons ici à un problème de la famille des problèmes d'accord : le consensus. Dans ce problème, tous les processus proposent une valeur, et tous les processus corrects doivent inéluctablement décider (propriété de terminaison) la même valeur (propriété d'accord), qui doit être une valeur proposée (propriété de validité). Ce problème, dont la définition est particulièrement simple, est un problème fondamental de l'informatique répartie puisqu'il abstrait beaucoup de problèmes d'accord. Comme nous l'avons déjà vu, [FLP85] prouve que ce problème n'a pas de solution dans un système asynchrone dès que le moindre processus peut crasher. Il faut alors enrichir le modèle du système réparti avec des hypothèses supplémentaires pour pouvoir résoudre le consensus. Parmi les multiples types d'enrichissement proposés, citons l'ajout de synchronie [DLS88], l'abandon d'une solution déterministe pour une solution probabiliste [BCBT96, Rab83] et l'emploi de détecteurs de défaillances [CT96]. Ces enrichissements permettent de résoudre le consensus malgré l'asynchronie et les défaillances. Notons que dans le cas du consensus byzantin, nous utilisons la propriété de validité suivante : si tous les processus corrects proposent la même valeur v, alors seule v peut être décidée.

4.1.2 Travaux connexes

Nous avons vu qu'il fallait enrichir les systèmes asynchrones pour pouvoir résoudre le consensus en présence de crashs. Cette approche est abstraite par la définition de détecteurs de défaillances [CT96]. Un détecteur de défaillances est un oracle donnant des indications sur quels processus ont crashé. Ces indications peuvent être fausses. La quasi-totalité des implémentations des détecteurs de défaillances considère qu'inéluctablement le système se comporte de manière synchrone (*i.e.* l'asynchronie n'est que transitoire). Plus précisément, ils emploient des systèmes dits *partiellement synchrones* [CT96] qui sont une généralisation des modèles proposés dans [DLS88].

Les systèmes partiellement synchrones sont l'objet de la majorité des travaux sur le consensus byzantin [ADGFT06, CL99, KMMS97, DGG02, DGV05, MA06]. Les articles [KMMS97, DGG02, DGGS99] sont consacrés à la réalisation d'un détecteur de mutisme : il s'agit d'un détecteur de défaillance permettant de différencier un processus byzantin muet (ne répondant pas) d'un processus correct mais lent. Le protocole proposé dans [FMR05] utilise directement un détecteur de défaillances inéluctablement parfait. Les protocoles à la Paxos [BDFG03, MA06] établissent en premier lieu un leader avant de résoudre le consensus ou d'implémenter un automate. Enfin, [DGV05, Lam06] établissent des bornes basses par rapport à la tolérance aux défaillances et à la rapidité de décision. [DGV05] propose un algorithme générique paramétrable : avec ou sans authentification, décision rapide ou très rapide. Ces deux articles associent aux processus 3 rôles : les « proposeurs », les « apprenants » et les « accepteurs » (un processus peut jouer plusieur rôles).

Pour un système composé de n processus partiellement synchrones parmi lesquels au plus t peuvent crasher, de nombreux modèles [HMSZ55, ADGFT04] tentent de restreindre l'hypothèse de synchronie inéluctable à une sous-partie de l'ensemble des processus et du réseau de communication. Ceci différencie ces travaux des approches citées plus haut où tout le système était sensé remplir cette hypothèse. Ici, un lien est dit synchrone au temps τ si un message envoyé à l'instant τ est reçu avant l'instant $\tau + \delta$. La borne δ n'est pas connue et n'est valable qu'après un temps inconnu τ_{GST} (pour temps global de stabilisation, ou *Global Stabilization Time*). Un lien est dit inéluctablement synchrone s'il est synchrone pour tout $\tau \geq \tau_{GST}$. Le modèle proposé dans [ADGFT04] suppose qu'au moins un processus correct a t liens sortants inéluctablement synchrones. Un tel processus est appelé une $\diamond t$-source (le $\diamond$ dénote le caractère inéluctable). Le modèle considéré dans [HMSZ55] suppose un système avec une primitive de diffusion et au moins un processus correct ayant t liens bidirectionnels éventuellement synchrones, mais ces t liens peuvent

être « mobiles », c'est-à-dire que l'ensemble des t processus impliqués dans ces liens peut changer. Ces deux modèles ne sont pas comparables [HMSZ55]. Dans un tel modèle, [ADGFT04] prouve qu'une $\diamond t$-source est nécessaire et suffisante pour résoudre le problème.

Dans le contexte où les t processus peuvent exhiber un comportement byzantin, Aguilera et al. [ADGFT06] proposent un modèle avec de faibles hypothèses concernant la synchronie permettant de résoudre le consensus. Ce modèle suppose qu'il existe au moins un processus correct dont tous les liens sortants et entrants sont inéluctablement synchrones (on parle alors de $\diamond$ bisource). Leur protocole ne nécessite pas d'authentification, mais utilisent des procédure de communication très coûteuses : elles sont similaires au broadcast cohérent et au broadcast authentifié [ST84]. Leur protocole consiste en une série de rondes, chaque ronde comportant 10 étapes de communication et une complexité en messages de $\Omega(n^3)$. Enfin, notons qu'une autre approche consiste à construire un détecteur d'erreurs transitoires [BDDT07]: l'objectif est de détecter une source comportement byzantin afin de l'empêcher d'interférer ensuite avec le reste du système.

Dans cette partie, nous proposons un modèle où les processus sont inéluctablement synchrones et où les communications sont partiellement synchrones : nous supposons que seule une partie des liens du système est inéluctablement synchrone. Nous sommes ainsi dans un modèle plus fort que le modèle asynchrone mais plus faible que le modèle inéluctablement synchrone [DLS88] ou le consensus byzantin peut être résolu si $t < n/3$ (avec ou sans authentifications). Ce modèle est décrit par la notion de bisource inéluctable de taille x (*i.e.* x liens entrants et sortants inéluctablement synchrones). L'inéluctable bisource utilisée dans [ADGFT06] a une taille maximale ($x = n - 1$). Informellement, une x-bisource est un processus correct ayant x voisins privilégiés (*i.e.* avec des échanges bidirectionnels synchrones) au lieu de $n - 1$ pour la bisource traditionnelle.

Dans ce modèle avec authentification et une $\diamond 2t$-bisource nous proposons un protocole résolvant le consensus byzantin. Pour cela nous supposons que le nombre de processus byzantins t est strictement inférieur à $n/3$, ce qui est la borne basse du consensus byzantin [DLS88]. Le protocole proposé est très simple en comparaison aux autres protocoles à la Paxos. De plus, dans les bonnes configurations la décision est prise en 5 étapes de communications indépendamment du comportement des byzantins. Une bonne configuration est une configuration où le premier coordinateur (ce rôle sera précisé par la suite) est une $\diamond 2t$-bisource.

Cette propriété est très intéressante puisque dans les configurations normales, la plupart des systèmes ont des comportement synchrones (les chances que le premier coordinateur soit une $\diamond 2t$-bisource sont ainsi fortes) et les pannes sont rares (le premier coordinateur est souvent correct). Notre protocole termine dès qu'une partie du système possède les propriétés d'une $\diamond 2t$-bisource. Dans ce sens le protocole « attend » que le réseau de communication sous-jacent ait la bonne structure (celle qui permet au coordinateur d'imposer sa valeur à suffisamment d'autres processus) avec la garantie de ne pas décider à tort si cette structure n'est pas présente. Lorsqu'elle se présente, la structure assure la bonne diffusion de la valeur du coordinateur, ce qui permet aux noeuds corrects de décider en tout sécurité.

4.1.3 Modèle de calcul et définition du problème

Modèle

Le modèle utilisé est inspiré du modèle partiellement synchrone décrit dans [DLS88]. Il est constitué d'un ensemble fini Π de n processus $(n > 1)$ complètement connectés. On pose $\Pi = \{p_1, \ldots, p_n\}$. Jusqu'à t processus peuvent exhiber un comportement *byzantin*. Un tel processus peut se comporter de manière arbitraire, indépendamment du code et du comportement prévu initialement par le concepteur du système. Il s'agit à ce titre du pire modèle de défaillance : un byzantin peut tomber en panne et s'arrêter, commencer dans un état arbitraire, envoyer des valeurs

différentes à différents noeuds, etc. Un processus ayant un comportement byzantin est dit *crashé*. Sinon il est *correct*.

Le réseau de communication est fiable dans le sens où un message envoyé par un processus correct à un autre processus correct est reçu une unique fois en un temps fini. Les messages ne sont pas altérés et le récepteur connaît l'identité de l'émetteur : nous utilisons des liens authentifiés asynchrones. D'après [HMSZ55], en utilisant certaines techniques [ACK+97, BCBT96], un tel réseau de communication peut être construit à partir d'un réseau constitués de liens « fair lossy », c'est-à-dire des liens ayant des pertes mais où un message envoyé une infinité de fois sera reçu une infinité de fois.

Chaque processus exécute un algorithme constitué d'une série d'étapes atomiques (envoyer un message, recevoir un message, effectuer un calcul local). Nous supposons que les processus sont partiellement synchrones : chaque processus correct effectue au moins une étape toutes les θ étapes du processus le plus rapide (θ n'est pas connu). Comme dans [DLS88], le temps est mesuré en nombre d'étapes effectuées par le processus le plus rapide. En particulier, la borne (inconnue) sur les temps de transfert des messages, δ, est telle que n'importe quel processus peut effectuer au plus δ étapes alors qu'un message synchrone est en transit. De cette façon, nous pouvons utiliser le comptage d'étapes locales comme mesure de temps pour l'attente des messages. Nous définissons maintenant formellement la bisource (à partir de la définition de [KMMS97]) :

Definition 1 *Un lien d'un processus p a un processus q est dit synchrone au temps τ si $1)$ aucun message envoyé par p a l'instant τ n'est reçu par q après l'instant $\tau + \delta$, ou $2)$ si q n'est pas correct.*

Definition 2 *Un processus p est une x-bisource à l'instant τ si*

- *p est correct*
- *Il existe un ensemble X contenant x processus tel que pour tout processus q de X les liens de p vers q et de q vers p sont synchrones à l'instant τ. Les processus de X sont dits voisins privilégiés de p.*

Definition 3 *Un processus p est une $\diamond x$-bisource s'il existe un instant τ tel que pour tout instant $\tau' \geq \tau$, p est une x-bisource à l'instant τ'.*

Notre modèle suppose l'existence de moyens d'authentification. Un processus byzantin peut disséminer une valeur fausse (*i.e.* différente de celle qu'il aurait envoyé s'il était correct). Afin de prévenir une telle dissémination, le protocole utilise des certificats : ce sont des signatures au niveau applicatif (de type cryptographie à clé publique, comme les signatures RSA). Une implémentation simple consiste à inclure un ensemble de messages signés comme certificat. Prenons le cas d'un processus p souhaitant relayer la valeur v reçue d'un processus q. Le processus q signe son message et l'envoie à p. Le processus p peut prétendre ne pas avoir reçu la valeur de q (ce n'est pas vérifiable) mais s'il relaie une valeur, c'est nécessairement la valeur signée par q. Ceci suppose que nos processus byzantins ne sont pas capables de falsifier les signatures. Supposons maintenant que p doive relayer à tous les processus la valeur qu'il a reçue majoritairement (parmi toutes les valeurs reçues). Le certificat que p joint consiste en l'ensemble des messages signés qu'il a reçu (de manière à ce que chaque processus puisse vérifier que p a bien envoyé la valeur majoritaire).

Un certificat pour un message m envoyé par p contient au moins $(n - t)$ messages reçus par p, tels que ces messages ont amené p à envoyer m selon le protocole. Ces certificats n'empêchent pas le comportement byzantin : comme dans beaucoup de systèmes asynchrones, lors des échanges généralisés (tout le monde échange avec tout le monde), un processus attend au plus $n-t$ messages (sinon il risque de se bloquer indéfiniment, voir section 3.1.1). Deux ensembles différents de $n - t$

valeurs peuvent contenir différentes valeurs majoritaires (majorité relative), chacune certifiable. Ainsi un processus byzantin recevant plus de $(n-t)$ messages peut envoyer différentes valeurs majoritaires certifiées à différents processus.

Le consensus

Nous avons présenté informellement le consensus dans l'introduction. Nous considérons la version multivaluée du consensus : les processus peuvent proposer des valeurs v d'un ensemble V dont le cardinal n'est pas borné (par opposition au consensus binaire où $|V| = 2$). Chaque processus p_i *propose* une valeur v_i et tous les processus corrects doivent inéluctablement *décider* une valeur v en relation avec l'ensemble des valeurs proposées. Notons qu'il est nécessaire d'adapter la définition du problème au contexte byzantin : il est par exemple impossible d'exiger qu'un processus byzantin décide une valeur puisqu'il agit arbitrairement et donc décide ce qu'il veut. De même, la notion de « valeur proposée » n'a plus de sens puisqu'un processus byzantin peut proposer des valeurs différentes à des processus différents. Ainsi, nous définissons le problème du consensus dans ce contexte par les trois propriétés suivantes :

Terminaison Tous les processus corrects décident inéluctablement

Accord Tous les processus corrects décident la même valeur

Validité Si tous les processus corrects proposent la même valeur v, alors seule v peut être décidée.

4.1.4 Le protocole

Nous allons ici présenter le protocole dont le code est donné à la figure 4.1. Il est composé de 4 phases dont nous allons donner l'intuition.

Chaque processus p_i maintient une variable locale est_i qui contient l'estimation courante de la valeur décidée. La phase d'initialisation (lignes 1-3) consiste en une phase d'échange globale (tous vers tous) permettant aux processus de positionner est_i à une valeur qu'ils auraient éventuellement reçue $(n-2t)$ fois. Cette phase n'utilise pas les certificats puisque c'est la première phase : il n'y a pas de communication antérieure à prouver. Si p_i n'a pas reçu $(n-2t)$ fois la même valeur, il initialise est_i à v_i, la valeur qu'il propose. Cette phase permet d'établir la propriété de validité du consensus puisque si tous les processus corrects proposent la même valeur v, tous le processus vont recevoir cette valeur au minimum $(n-2t)$ fois, et v sera la seule valeur pouvant être reçue $n-2t$ fois et donc la seule valeur certifiée. A partir de la ligne 5, tous les messages échangés à chaque phase sont signés, et incluent comme certificat $(n-t)$ messages reçus par le processus à la phase précédente.

Concernant la validité des messages, nous supposons que chaque processus possède un démon filtrant les messages reçus, par exemple les messages dupliqués (nécessairement issus de processus byzantins puisque les autres processus sont corrects et sont reliés par des canaux fiables). Le démon filtre aussi les messages dont la syntaxe n'est pas correcte, ainsi que les messages dont le certificat n'est pas valide ou ne concorde pas.

Après la phase d'initiation, le protocole est constitué d'une séquence de rondes consécutives. Chaque processus maintient une variable locale r_i initialisée à 0. Chaque ronde est coordonnée par un processus p_c prédéterminé (par exemple, $c = r \mod n$). Cette technique dite du *coordinateur tournant* est classique. Chaque ronde est constituée de 4 phases de communication.

La première phase d'une ronde r est décrite par les lignes $5-7$. Chaque processus démarrant une ronde (y compris le coordinateur) envoie en premier son estimation (avec le certificat associé) au coordinateur (p_c) de la ronde courante à l'aide d'un message `Query` et arme un temporisateur à la durée $\Delta_i[c]$. Δ_i est un tableau de durées maintenues par p_i : lorsqu'un temporisateur expire avant la réception par p_i de la réponse d'un processus p_j, $\Delta_i[j]$ est incrémenté. Cela permet

d'inéluctablement atteindre la borne haute de temps de transit d'un message entre p_i et p_j s'ils sont des voisins privilégiés. De plus, ceci évite que p_i se bloque en attendant (ligne 6) le message d'un coordinateur crashé. Lorsque le coordinateur d'une ronde r reçoit un message `Query` valide contenant une estimation est pour la première fois à la ligne 19, il envoie un message `Coord`(r, est) à tous les processus.

Le message `Coord` est envoyé à partir d'une autre tâche parallèle puisque le coordinateur c d'une ronde r pourrait être bloqué dans une ronde antérieure. Le temps pour c d'arriver à la bonne ronde et le temporisateur du processus ayant émis le message `Query` aura expiré. Ainsi le processus coordonnant une ronde réagit dès qu'il reçoit la requête associée, même s'il est lui-même encore dans une ronde précédente. Le processus coordinateur réagit en émettant la valeur estimée reçue dans la requête à tous les processus du système.

Le coordinateur peut exhiber trois comportements :

- Si le coordinateur est une $2t$-bisource, il a $2t$ voisins privilégiés parmi lesquels au moins t processus sont corrects. Le cas échéant, il y a donc $(t+1)$ processus corrects en possession de la valeur du coordinateur (les t voisins corrects privilégiés et le coordinateur lui-même). Ces processus positionnent leur variable aux à v (avec $v \neq \perp$, $\perp$ étant la valeur par défaut, qui n'est pas proposable).
- Si au contraire le coordinateur est byzantin, il peut soit ne rien envoyer, soit envoyer différentes valeurs certifiées à différents processus (dans un tel cas nous verrons qu'il faut qu'aucune de ces valeurs n'ait été décidée à la ronde précédente).
- Si le coordinateur n'est ni byzantin, ni une bisource suffisante, les phases suivantes du protocole assurent que les processus corrects du système se comportent de manière cohérente. Soit aucun processus ne décide, soit certains décident une valeur v, mais alors ce sera la seule valeur certifiable aux rondes suivantes, interdisant aux byzantins l'introduction de nouvelles valeurs.

La deuxième phase d'une ronde r (lignes $8-10$) tente d'étendre la portée de la $2t$-bisource. Dans cette phase, tous les processus relaient la valeur certifiée obtenue du coordinateur, ou $\perp$ s'ils n'ont rien reçu du coordinateur. Chaque processus attend au moins $(n-t)$ messages valides (dont la valeur est stockée dans l'ensemble V_i). Si le coordinateur est une $2t$-bisource, alors au moins $(t+1)$ processus corrects possèdent la valeur du coordinateur : tous les processus corrects sont donc maintenant en possession de la valeur du coordinateur dans ce cas (puisque chaque processus manque au maximum t messages). Cette phase n'a pas d'effet sinon. La condition $(V_i - \{\perp\} = \{v\})$ de la ligne 10 sert à s'assurer qu'il n'existe plus qu'une seule valeur non-$\perp$ dans V_i. Cette valeur est alors stockée dans aux_i.

La troisième phase d'une ronde r (lignes $11-13$) n'a pas d'effet si le coordinateur est correct. Cette phase est utile dans le cas d'un coordinateur byzantin : dans ce cas différents processus corrects peuvent avoir leur variable aux à différentes valeurs (puisque le coordinateur byzantin peut certifier différentes valeurs). La phase 3 est un filtre, puisqu'à la fin de cette phase, au plus une valeur non-$\perp$ peut être conservée dans les variables aux. Cette phase consiste en un échange global, où chaque processus collecte au minimum $(n-t)$ messages. Si tous ces messages contiennent la même valeur v alors $aux_i = v$, sinon $aux_i = \perp$.

La quatrième phase d'une ronde r est la phase de décision (lignes $14-17$). L'objectif est d'assurer que la propriété d'accord n'est jamais violée. A cette phase, si un processus correct décide v alors v sera la seule valeur certifiable du système pour les rondes suivantes. Après un échange global, chaque processus collecte à nouveau au minimum $(n-t)$ messages valides qu'il stocke dans V_i. Si l'ensemble V_i contient $(n-t)$ fois la valeur v, alors p_i décide v. En effet, parmi ces $(n-t)$ valeurs, $t+1$ sont envoyées par des processus corrects. Puisqu'il n'existe plus qu'une valeur certifiable, cela veut dire que tous les processus corrects ont reçu au moins une fois la valeur v. Avant de décider cependant, le processus envoie un message signé `DEC` contenant la

Function Consensus(v_i)

Init: $r_i \leftarrow 0$; $\Delta_i[1..n] \leftarrow 1$;

Task $T1$: % basic task %

——————————————————- init phase ——————————————————

1: *send* INIT(v_i) to all;
2: **wait until** (INIT messages received from at least $(n-t)$ distinct processes);
3: **if** ($\exists v$: received at least $(n-2t)$ times) **then** $est_i \leftarrow v$ **else** $est_i \leftarrow v_i$ **endif**;

repeat forever
4: $c \leftarrow (r_i \mod n) + 1$; $r_i \leftarrow r_i + 1$;

——————————————————- round r_i ——————————————————

5: *send* QUERY(r_i, est_i) to p_c; *set_timer*($\Delta_i[c]$);
6: **wait until** (COORD(r_i, est) received from p_c or *time-out*) **store value** in aux_i; *% else* $\perp$ *%*
7: **if** (*timer times out*)) **then** $\Delta_i[c] \leftarrow \Delta_i[c] + 1$ **else** *disable_timer* **endif**;

8: *send* RELAY(r_i, aux_i) to all;
9: **wait until** (RELAY($r_i, *$) received from at least $(n-t)$ distinct processes) **store values** in V_i;
10: **if** ($V_i - \{\perp\} = \{v\}$) **then** $aux_i \leftarrow v$ **else** $aux_i \leftarrow \perp$ **endif**;

11: *send* FILT1(r_i, aux_i) to all;
12: **wait until** (FILT1($r_i, *$) received from at least $(n-t)$ distinct processes) **store values** in V_i;
13: **if** ($V_i = \{v\}$) **then** $aux_i \leftarrow v$ **else** $aux_i \leftarrow \perp$ **endif**;

14: *send* FILT2(r_i, aux_i) to all;
15: **wait until** (FILT2($r_i, *$) received from at least $(n-t)$ distinct processes) **store values** in V_i;
16: **case** ($V_i = \{v\}$) **then** *send* DEC(v) to all; **return**(v);
17: ($V_i = \{v, \perp\}$) **then** $est_i \leftarrow v$;
18: **endcase**;

———————————————————————————————————————

end repeat

Task $T2$: % coordination task %
19: **upon** *receipt* of QUERY(r, est) for the first time for round r: *send* COORD(r, est) to all;

Task $T3$:
20: **upon** *receipt* of DEC(est): *send* DEC(est) to all; **return**(est);

FIG. 4.1 – Protocole de consensus byzantin (suppose une $\diamond 2t$-bisource)

valeur décidée et son certificat. Cela évite aux autres processus de se bloquer à la ronde suivante parce que certains processus ont déjà décidé. Lorsqu'un processus reçoit un message DEC à la ligne 20, il le relaie à tous les autres et décide. L'envoi des messages DEC peut être vue comme une implémentation du *reliable broadcast*.

4.1.5 Correction du protocole

Nous allons ici prouver formellement les propriétés du protocole. Nous appellerons échange global la combinaison d'un envoi généralisé (envoi à tous les processus du système), suivi d'une collecte par tous les processus. Chaque processus correct collecte $(n-t)$ messages puisque seuls les processus byzantin peuvent être muets. Il y a 4 échanges globaux : lignes $1-2$, $8-9$, $11-12$ et $14-15$.

Lemme 1 *Soit V_i et V_j deux ensembles collectés par deux processus corrects p_i et p_j après un*

échange global. Nécessairement on a

$$V_i \cap V_j \neq \emptyset$$

Preuve Par contradiction, on suppose $V_i \cap V_j = \emptyset$. Soit S l'ensemble des messages envoyés à p_i et p_j. On a $|S| = |V_i| + |V_j|$. Donc $|S| = 2 \times (n-t)$ puisque chaque processus attend au moins $(n-t)$ messages. Soit f le nombre effectif de byzantins ($f \leq t$). On a donc $n-f$ processus corrects envoyant nécessairement le même message à p_i et p_j et f processus byzantins pouvant envoyer des messages différents à nos deux processus.

On a donc $|S| \leq (n-f) + 2 \times f = (n+f)$ et ainsi $|S| \leq (n+t)$. Or $|S| \geq 2 \times (n-t)$, d'où $(n+t) \geq 2 \times (n-t)$, d'où $n \leq 3t$ ce qui est une contradiction puisque $n > 3t$. $\square_{Lemme}$

Lemme 2 *Après l'échange global lignes* $11-12$*, au plus une valeur non-*$\perp$ *peut être certifiée.*

Preuve Soit une exécution où un processus p_i collecte seulement des messages contenant la valeur v. Le processus p_i garde la valeur v, et les messages collectés sont le certificat de v. D'après le lemme 1, deux ensembles collectés s'intersectent forcément, donc il n'est pas possible d'exhiber un autre ensemble de $(n-t)$ valeurs w : aucune autre valeur ne peut être certifiée. $\square_{Lemme}$

Corollaire 3 *Si un processus décide* v *durant une ronde, alors seule la valeur* v *peut être décidée dans la ronde ou les rondes suivantes.*

Preuve Considérons le premier échange de message qui a mené p_i à décider une valeur certifiée v. p_i n'a donc reçu que des valeurs v certifiées. D'après le lemme 2, v est la seule valeur certifiée, donc les messages transportent soit v soit $\perp$. D'après le lemme 1, on a $\forall j, V_i \cap V_j \neq \emptyset$. Puisque p_i n'a reçu que des v, nécessairement chaque processus correct a reçu au moins une valeur v. Si ce processus décide, alors il décide v, sinon v sera la seule valeur certifiable pour les prochaines rondes. $\square_{Corollaire}$

Théorème 9 *Accord : Il n'existe pas deux processus décidant différemment.*

Preuve Si un processus décide ligne 20, il décide une valeur certifiée déjà décidée par un autre processus. Considérons la première ronde où un processus décide ligne 16. D'après le corollaire 3, si un processus décide durant la même ronde, alors il décide la même valeur. La seule valeur justifiable pour les rondes suivantes est v, donc tous les processus décident v. $\square_{Théorème}$

Lemme 3 *Si aucun processus ne décide durant une ronde* $r' \leq r$ *alors tous les processus commencent la ronde* $r+1$.

Preuve Observons d'abord que grâce aux temporisateurs, aucun processus correct ne peut rester bloqué dans la phase initiale.

Par contradiction, supposons qu'aucun processus ne décide à une ronde $r' \leq r$, où r est le plus petit numéro de ronde où un processus correct p_i bloque indéfiniment. Le processus p_i est bloqué à la ligne 9,12 ou 15.

Si p_i est bloqué à la ligne 9, première ligne où un processus correct peut bloquer, pendant la première ronde où un processus bloque, cela veut dire que tous les processus corrects ont exécuté la ligne 8. Il y a donc au moins $(n-t)$ messages d'envoyés. Puisque la communication est fiable, les messages envoyés arrivent inéluctablement, donc p_i reçoit inévitablement $(n-t)$ messages, donc n'est pas bloqué.

Si p_i est bloqué à la ligne 12, de la même manière puisque aucun processus correct ne bloque à la ligne 9, tous les processus exécutent la ligne 11, donc p_i ne peut pas bloquer à la ligne 12. Idem pour la ligne 15. Donc p_i n'est pas bloqué et s'il ne décide pas, il commence la ronde suivante. $\square_{Lemme}$

Théorème 10 *Terminaison : S'il existe une $\diamond 2t$-bisource dans le système, alors tous les processus corrects décident inéluctablement.*

Preuve Si un processus correct décide, grâce aux messages `DEC` de la ligne 16, tous les processus corrects vont recevoir ce message et décider (ligne 20).

Supposons donc qu'aucun processus ne décide. Par hypothèse, il existe un temps τ après lequel il existe un processus p_x qui est une $2t$-bisource. Puisque aucun processus ne décide, le temps du temporisateur est continuellement augmenté (ligne 7), jusqu'à dépasser la borne du temps de communication du système. Il existe donc un temps τ' après lequel les temporisateurs des voisins privilégiés de p_x n'expirent plus. Soit r la première ronde après τ' qui est coordonnée par p_x. Par hypothèse puisque aucun processus ne décide et d'après le lemme 3, tous les processus corrects commencent la ronde r.

Tous les processus corrects (plus peut-être des byzantins) commençant la ronde r envoient un message `Query` à p_x (ligne 5). Lorsque le coordinateur p_x de la ronde r reçoit le premier message (ligne 19), il envoie un message `Coord`. Si l'on considère un processus correct p_i voisin privilégié de p_x, ce message `Coord` est envoyé au plus tard lorsque p_x reçoit le message `Query` de p_i, donc aucun voisin privilégié correct de p_x ne rate le message de p_x.

Dans le pire cas, il y a les t byzantins parmi les $2t + 1$ voisins privilégiés de p_x. La valeur v de p_x est donc relayé au moins $t + 1$ fois (ligne 8) par ses voisins corrects privilégiés. Puisque chaque processus collecte $(n - t)$ messages au minimum (donc en perd au maximum t), la valeur du coordinateur est disséminée à tous les processus. Il est important d'insister ici sur le fait que même les processus byzantins ne peuvent pas mentir sur le fait qu'ils ont reçu la valeur de p_x ligne 10 puisque tout ensemble de $n - t$ messages contient au moins une valeur v.

Au cours de la troisième phase (ligne $11-13$), tous les processus émettant un message émettent v (les byzantins peuvent rester muets ou envoyer la valeur v). Donc tous les processus ont leur variable aux à v ligne 13. Ainsi tous les processus émettant un message certifié émettent v ligne 14. On peut donc conclure que tous les processus corrects décident v à la ligne 16. $\square_{Théorème}$

Théorème 11 *Validité : si tous les processus proposent v, alors seule v peut être décidée.*

Preuve Soit v la seule valeur proposée par les processus corrects. v est donc envoyée au moins $(n - t)$ fois ligne 1. Tous les processus reçoivent donc v au moins $(n - 2t)$ fois ligne 3. Les byzantins peuvent au pire proposer la même valeur t fois et, puisque $n > 3t$, $t < n - 2t$ donc v est la seule valeur certifiée. $\square_{Théorème}$

4.1.6 Remarques

Nous allons maintenant faire quelques remarques sur le protocole présenté ci-dessus.

Minimalité

Un processus attendant la valeur d'un coordinateur vivant mais possiblement byzantin est face à un dilemme : attendre ou ne pas attendre. S'il attend mais que le coordinateur est byzantin, cela peut bloquer le protocole. S'il n'attend pas le coordinateur qui est une bisource lente, le

protocole peut ne pas décider. Ce dilemme est similaire à celui rencontré dans les systèmes asynchrones soumis aux crashs (voir 3.1.1). La seule façon de sortir de ce dilemme est de supposer qu'inéluctablement le coordinateur a des liens synchrones entrants.

Le nombre de liens inéluctablement synchrones est au moins $2t$. S'il y a moins de liens, dans le pire scénario seuls t processus possèdent la valeur du coordinateur, et par conséquent les $n-t$ autres processus corrects du système peuvent rater indéfiniment la valeur du coordinateur puisque chaque processus n'attend que $n-t$ messages.

C'est pourquoi nous pensons qu'une $\diamond 2x$-bisource est nécessaire si $x \geq t$ processus peuvent présenter un comportement byzantin. De plus, nous pensons que $1)$ pour atteindre cette borne l'authentification est nécessaire et $2)$ sans authentification il faut une $\diamond 3x$-bisource si $x \geq t$.

Structure du réseau

Du point de vue de la structure du réseau de communication, ce protocole est très intéressant. En effet son comportement consiste à attendre l'apparition d'une structure permettant la diffusion suffisante de la valeur du coordinateur, et ensuite de s'assurer de l'efficacité de cette diffusion, rien de plus.

Sous cet aspect, la structure que nous étudions est différente des structures précédemment abordées dans cette thèse : il s'agit de structures évoluant à travers le temps. Ce dynamisme impose de nouvelles contraintes dans la conception du protocole. Il ne s'agit en effet plus simplement de trouver la bonne paire (hypothèses, protocole), mais il s'agit aussi de résister à l'absence des conditions de terminaison (c'est-à-dire aux périodes d'asynchronie) tout en conservant un système cohérent.

Il est aussi intéressant de remarquer que la structure qui nous intéresse ici n'est pas exactement le maillage des liens synchrones, mais plutôt le maillage des échanges de valeurs. Puisque nous utilisons lors du relai des valeurs un système de "réponses gagnantes" (*i.e.* le processus attend les $n-t$ premières réponses), dans certaines exécutions il est possible d'observer des échanges de valeurs n'empruntant pas les liens synchrones. Il est même possible et probable que ces échanges de valeurs permettent au protocole de terminer. Ceci amène deux constats:

- Les contraintes énoncées sur la structure sont les contraintes suffisantes pour permettre au protocole de terminer quelle que soit l'exécution. Ce ne sont pas les contraintes nécessaires pour chaque exécution.
- Le recours à l'hypothèse d'une $2t$-bisource résume deux hypothèses qui sont « la valeur du coordinateur est suffisamment répandue » et « aucun processus ne nie avoir reçu la valeur du coordinateur ». C'est finalement ces deux propriétés qui sont fondamentales pour la résolution du protocole, et les hypothèses sur la synchronie du système sont le moyen le plus pratique d'exprimer indirectement ces propriétés.

En résumé, ce protocole utilise une propriété de la structure de communications pour terminer, tout en assurant qu'une structure moins robuste que nécessaire n'engendre pas de problèmes de cohérence.

4.2 Établissement de secret dans les réseaux de capteurs

Nous allons dans cette partie présenter une exploitation répartie des particularités de la structure de communications permettant d'établir un secret (*i.e.* une clé) partagé par certaines paires d'entités du système. Ce secret peut ensuite servir à l'établissement de communications chiffrées entre les entités concernées.

4.2.1 Motivation

La structure des communications dans les réseaux de capteurs est basée sur le voisinage. Comme présenté dans l'introduction et dans le chapitre précédent, ici les entités ne sont plus discriminée par la vitesse de leurs échanges mais sur les acteurs de ces échanges : chaque entité ne peut communiquer qu'avec ses voisins proches.

Les capteurs utilisent pour communiquer les ondes radio. Ce médium est problématique dès lors qu'il s'agit d'établir une confidentialité puisque la propagation des ondes radio n'est pas contrainte. Ainsi un noeud émettant est entendu par tous ses voisins. Il est difficile partant de ce constat d'établir un secret entre deux capteurs voisins.

De plus, l'utilisation classique des réseaux de capteurs est la surveillance d'un paramètre physique sur une zone géographique donnée. Les données issues de cette surveillance sont acheminées de proche en proche jusqu'à un noeud chargé de leur centralisation, souvent appelé *puits* ou *sink*. A cette fin, chaque message est relayé par plusieurs capteurs intermédiaires (on parle de communication *multi-hop* ou multi-sauts). Chaque intermédiaire est une menace pour la confidentialité du message et pour son intégrité : il peut facilement connaître et/ou modifier le message qu'il relaie.

Enfin, les capteurs sont des entités dont les capacités et le déploiement impose de nombreuses contraintes. Déployés dans des environnements hostiles, l'objectif est de limiter au maximum l'intervention humaine nécessaire à l'établissement d'un réseau efficace. Ceci passe par l'utilisation d'algorithmes répartis aux propriétés auto-organisantes. Ces capteurs sont produits en masse, il est donc difficile et coûteux de les préconfigurer de manière précise. Enfin, ces entités sont limitées en capacité de stockage et de calcul et en autonomie. Il est donc important de concevoir des algorithmes légers en temps de calcul et nécessitant peu de communication.

4.2.2 Travaux connexes

La sécurisation des réseaux de capteurs et particulièrement l'établissement de secrets partagés, est un problème difficile du fait des contraintes énumérées ci-dessus. Il est principalement abordé selon deux axes : utiliser de la cryptographie asymétrique, et utiliser de la cryptographie symétrique basée sur des clés réparties aléatoirement aux capteurs avant leur déploiement.

L'utilisation de la cryptographie asymétrique est sujette à controverses. Jusque récemment, elle était considérée comme trop onéreuse en temps de calcul pour être utilisée dans les capteurs. Par exemple, [CKM00] reporte que sur un capteur de type Motorolla « Dragonball », chiffrer avec des clés de 1024 bits en RSA coûte 42mJ, contre 0.104 mJ pour une même taille de clés en AES. Récemment cependant, l'arrivé de la cryptographie à courbe elliptique a changé la donne. Celle-ci ne repose en effet plus sur l'exponentiation, opération coûteuse en temps de calcul, si bien que les expériences d'utilisation de cryptographie asymétrique dans les réseaux de capteurs se multiplient (p.ex. [SOS+08]). Pour utiliser ces solutions, il reste néanmoins à mettre en place une solution efficace d'obtention de clés publiques (ou *Public Key Infrastructure*).

L'utilisation de la cryptographie symétrique dans les réseaux de capteurs a été largement explorée [EG02, APS03, ZXSJ03]. Efficace énergétiquement, elle nécessite cependant elle aussi un système de gestion de clés.

Les deux solutions extrêmes consistent soit à établir une clé pour chaque paire de capteurs soit à utiliser une unique clé pour tout le réseau. Dans le premier cas, c'est une sécurité très robuste mais coûteuse à mettre en place : il faut $o(n^2)$ espace mémoire pour n capteurs. Dans le second cas, chaque capteur ne stocke qu'une clé, mais si un adversaire découvre cette clé (par exemple en s'emparant d'un capteur), c'est tout le réseau qui est compromis. Les systèmes de gestion de clés actuels cherchent un compromis entre ces deux approches.

L'idée de base est de générer un grand ensemble de clés et d'attribuer une partie de ces clés à chaque capteur avant son déploiement. Selon le nombre de clés et la taille de l'ensemble de

clés attribuées à chaque capteur, deux capteurs du réseau peuvent utiliser des clés qu'ils ont en commun pour crypter leurs échanges. Il est possible d'illustrer ce principe à l'aide du « paradoxe des anniversaires » : dans un groupe de 23 personnes, il y a une probabilité $1/2$ de trouver deux personnes nées le même jour de l'année. C'est ce principe qui est utilisé ici : on donne aux capteurs un certain nombre de « dates » (les clés), et la communication cryptée est possible s'ils partagent au moins une date.

Les travaux existants dans ce domaine portent sur les méthodes de distribution des clés (comme par exemple [CA06]) et sur les méthodes de découverte des clés des autres capteurs [DPMM06]. L'objectif des travaux cités est de rendre le réseau résistant à un adversaire capturant des capteurs et apprenant ainsi certaines clés dans le but de compromettre les échanges du reste du réseau. Le défaut principal de ces méthodes est le recours à des préconfigurations dont la mise en oeuvre pratique n'est pas toujours aisée.

Dans cette section, nous présentons une nouvelle approche permettant l'établissement d'un secret entre certains couples de capteurs voisins. Mise à part les identités, les capteurs sont tous strictement identiques. Afin de créer des secrets partagés nous utilisons, à la place d'une préconfiguration difficile, la différence naturelle qui survient entre les capteurs au moment de leur déploiement : leur position dans la structure de communication. Plus précisément, notre algorithme repose sur l'observation que de nombreux couples de noeuds voisins ont une intersection de voisinage unique, ce qui permet de construire simplement des clés.

Ensuite, nous utilisons les communications de voisinage à présent sûres pour permettre à tous les noeuds du système de communiquer de manière confidentielle. Bien sûr, la structure du réseau de communication étant imposée par la disposition géographique des capteurs, il n'est pas possible de sécuriser tous les liens. Nous montrons à l'aide de simulations que, selon certaines distributions de noeuds, plus de 85% des noeuds peuvent communiquer de façon sécurisée.

4.2.3 Modèle

Nous présentons ici le modèle de calcul employé dans cette section. Nous présenterons la modèle de communication, le modèle d'adversité qui est une version affaiblie du byzantin que nous avons introduit précédemment, ainsi qu'une définition formelle du problème que nous souhaitons résoudre.

Communications

Nous utilisons un modèle proche de [OW05]. Un capteur est équipé d'un dispositif de communication radio lui permettant d'émettre et de recevoir des messages dans une certaine portée (nous supposons que cette portée est de 1). Notre algorithme requiert simplement des communications symétriques, mais pour la clarté de la présentation, nous modélisons la zone de portée radio par un disque. La structure des communications multisaut peut être représentée par un graphe $G = (V, E)$ où V est l'ensemble des capteurs et E l'ensemble des arêtes. Il existe une arête entre deux capteurs u et v si u et v sont à portée radio (*i.e.* ils peuvent communiquer directement). On dit alors qu'ils sont à une distance de 1 saut, ou voisins.

Nous considérons que les capteurs sont disposés dans un plan à deux dimensions. Ceci fait de G un graphe dit *UDG* pour Unit Disc Graph [CCJ90] : deux capteurs sont connectés si et seulement si leur distance est inférieure à 1.

La communication entre deux capteurs i et j est dite à k sauts si le plus court chemin entre les sommets i et j dans le graphe G est de taille k. Ceci implique qu'un message émis par i pour j devra transiter par $k - 1$ capteurs avant d'atteindre sa destination.

Nous considérons que les capteurs sont immobiles et qu'ils n'ont pas de notion de distance ni de position (pas de GPS). Nous considérons de plus un réseau connexe (il existe un chemin reliant

toute paire de sommets du graphe).

Rapport au temps

Pour découvrir son voisinage, un capteur envoie un message et attend une réponse de chacun de ses voisins. Comme il ne sait pas combien de voisins il possède, il ne sait pas combien de temps attendre. Pour borner cette attente, il faut borner dans le temps les communications : le système doit être synchrone.

Adversité

Le modèle traditionnel de défaillance pour les réseaux de capteurs reste le modèle « crash stop ». Nous avons déjà introduit les byzantins dans la partie précédente. Le contexte des réseaux de capteurs sans fil ajoute de nouvelles causes à l'apparition de byzantins :

involontaires A l'approche de la fin de sa batterie, ou à cause d'un problème de conception, ou d'un composant endommagé, un capteur peut avoir un comportement non spécifié [GGN09].

volontaires Un adversaire cherche à altérer le comportement du système. Cette altération peut prendre plusieurs formes. L'adversaire peut simplement détruire les capteurs, il peut aussi chercher à perturber le fonctionnement du système. Il peut ajouter des capteurs de sa conception. Enfin, il peut capturer un capteur et remplacer le code du capteur par un code de sa conception [TP07].

Le modèle général de l'adversaire byzantin conviendrait pour qualifier le comportement déviant de ces capteurs. Néanmoins, dans sa version originale [LSP82], l'adversaire byzantin est trop fort pour permettre quoi que se soit dans les réseaux de capteurs :

- S'ils n'ont pas de contraintes énergétiques, ils peuvent brouiller toutes les communications [GGN09].
- S'ils ont une capacité de calcul illimitée ils peuvent rendre inutiles les moyens cryptographiques déployés.
- Si le nombre de noeuds byzantins est trop élevé, ils peuvent déconnecter le réseau en refusant de transmettre des messages.

Nous proposons ici un modèle d'adversité plus faible, qui correspond plus aux attaques qu'un réseau peut subir. Nous supposons qu'une partie des noeuds peuvent exhiber un comportement byzantin (que ce soit suite à un ajout de noeuds modifiés ou à un piratage de noeuds présents). Nous supposons que les noeuds byzantins ne coopèrent pas, qu'ils ne brouillent pas les communications en créant des interférences et que leur capacité de calcul est bornée.

Dans ce sens, notre modèle est plus proche du modèle des participants d'un réseau pair à pair : chaque entité a sa volonté propre, décidée à utiliser le réseau pour profiter de ces fonctionnalités. Chaque noeud est un tricheur potentiel s'il a intérêt à le faire. Une image définissant bien l'adversaire et l'objectif est celle du jeu de poker : les capteurs participent à un jeu de poker. Le noeud puits distribue les cartes et fait office de tapis. Chaque noeud est byzantin dans le sens ou il cherche, par exemple, à deviner la main d'un autre noeud. Chaque noeud participe néanmoins, puisque s'il ne participe pas il ne gagne rien. Le but est de créer un protocole permettant cette application, sans que la main d'un noeud éloigné du puits soit dévoilée à tous les noeuds intermédiaires.

problème

La définition formelle du problème est la suivante : les capteurs démarrent dans un état où le seul paramètre les différenciant est leur identité. Nous voulons établir des liens protégés entre

noeuds voisins, c'est-à-dire procurer à chaque paire de capteurs voisins une clé secrète leur permettant d'utiliser de la cryptographie à clé publique pour communiquer de manière confidentielle (c'est-à-dire sans être intercepté par aucun autre capteur).

La solution au problème de l'établissement de clés secrètes partagées doit remplir trois conditions :

Complétude Chaque paire de noeuds voisins partage une clé

Symétrie La clé obtenue par u pour communiquer avec v est la même que celle obtenue par v pour communiquer avec u.

Confidentialité La clé obtenue par u et v n'est connue par aucun autre noeud.

4.2.4 Description de l'algorithme

Nous allons ici présenter un algorithme pour établir des communications confidentielles entre n'importe quelle paire de noeuds d'un réseau de capteurs. Pour cela, deux phases préliminaires sont nécessaires : l'établissement de liens sécurisés entre voisins, puis l'établissement de liens à deux sauts.

Etablissement de clés secrètes à un saut

L'objectif de cette phase est de permettre à un noeud d'établir des clés avec ses voisins, c'est-à-dire de permettre à un noeud de communiquer de façon confidentielle avec chacun de ses voisins. Un noeud a un ensemble fixé de voisins (on considère un réseau statique composé de noeuds immobiles), qui dépendent de la localisation physique du capteur.

Pour faciliter la compréhension, nous allons suivre l'établissement d'une clé à travers un exemple. La figure 4.2 représente une partie de réseau composée de 5 noeuds $\{A, B, C, D, E\}$. Ces noeuds diffèrent juste par leur identité. Nous voulons permettre au noeuds A et B d'établir une clé secrète commune leur permettant d'échanger de l'information sans que celle-ci soit déchiffrable par les noeuds C, D et E.

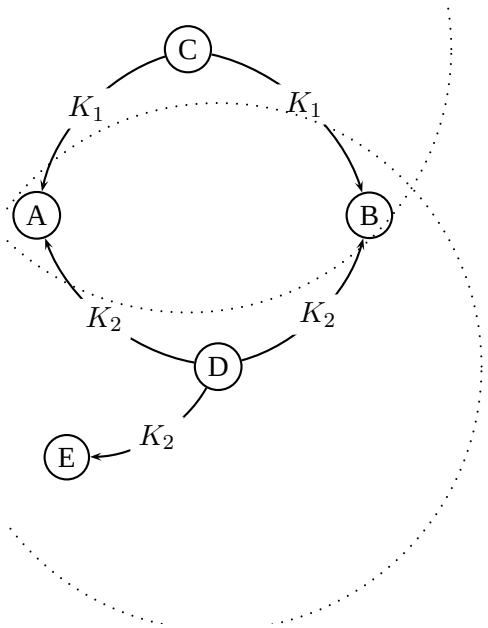


FIG. 4.2 – Exemple d'établissement de secret à un saut

(1) send(i)
(2) Let V_i be the set of received ids *% Set of neighbours including i itself*

(3) send(i, V_i)
(4) Let $W_i(j)$ be the set of ids received from sensor j forall $j \in V_i$ *% neighbours of the neighbours*

(5) Let $P_i = \{j \in V_i \setminus \{i\} | \cap_{k \in V_i \cap W_i(j)} W_i(k) = \{i, j\}\}$
(6) send(get_basic_key())
(7) Let $bk_i(k)$ be the basic key received from sensor k forall $k \in V_i$
(8) Forall $j \in P_i$, $key_i(j) \leftarrow$ comb_keys$(\{bk_i(k) | k \in V_i \cap W_i(j)\})$

FIG. 4.3 – Établissement des clés à un saut

A cette fin, observons l'ensemble des voisins communs à A et B. Il s'agit des noeuds C et D (E n'est pas voisin de B). Le principe de l'algorithme repose sur l'observation que puisque C et D ne sont pas voisins chacun peut procurer une moitié de la clé (K_1 et K_2 sur la figure 4.2) à A et B, et ainsi ces derniers seront les seuls à posséder la totalité de la clé $K_1 + K_2$.

Observons ce qui rend possible l'établissement d'une clé secrète entre A et B :

- Il faut au moins deux voisins communs à A et B.
- Il faut qu'aucun des voisins communs (sauf A et B) ne soit en mesure d'entendre la totalité des parties de clés procurées par les autres voisins communs. Autrement dit, il ne faut pas que tous les voisins communs soient mutuellement voisins.

De façon plus formelle, soit V_i l'ensemble des voisins du noeud i (y compris lui même). Nous définissons le *voisinage commun* à deux noeuds i et j comme $CN_{ij} = V_i \cap V_j$. Les deux observations précédentes se généralisent sous la condition

$$\textit{Cond}_{ij} : \bigcup_{k \in CN_{ij}} V_k = \{i, j\}.$$

Donc si deux voisins i et j satisfont la condition $\textit{Cond}_{ij}$, alors ils peuvent construire une clé secrète. Cette clé est une fonction des clés procurées par chaque noeud de CN_{ij}.

L'algorithme 4.3 présente le code exécuté par le noeud i. Il permet l'établissement de clés partagées entre i et ses voisins. Il s'agit d'un algorithme en trois étapes :

- La première phase (lignes 1 et 2) permet à i de découvrir son voisinage (V_i). Notons que nous supposons que i reçoit ses propres messages et les traite comme s'il s'agissait de messages issus de ses voisins (on pose donc $i \in V_i$).
- La deuxième phase (lignes 3 et 4) permet à i de découvrir les voisins de ses voisins (*i.e.* ses voisins à 2 sauts). La variable $W_i(j)$, locale à i, contient les voisins de j (*i.e.* V_j). Ligne 5, i calcule l'ensemble P_i qui est l'ensemble des noeuds de V_i avec lesquels il est possible d'établir une clé secrète partagée. Autrement dit, $j \in P_i \Leftrightarrow \textit{Cond}_{ij}$.
- Finalement, le noeud i envoie une clé aléatoire (ligne 6) afin que ses voisins puissent l'utiliser pour forger des clés eux aussi et reçoit les clés qui lui sont envoyées. On suppose que chaque noeud dispose d'une primitive get_basic_key() renvoyant une clé aléatoire d'une taille donnée. Le noeud i stocke les clés reçues dans la variable locale bk_i. Maintenant i est capable de forger les clés qu'il partage avec les noeuds de P_i. Pour chaque noeud $j \in P_i$, i calcule la clé secrète en combinant (grâce à un appel à la fonction comb_keys) les clés reçues de tous les noeuds de CN_{ij}. La combinaison de clés peut être un simple `xor`, ou toute autre opération déterministe.

Il est possible d'exprimer les trois propriétés de l'établissement de clés secrètes partagées avec les variables de l'algorithme :

Complétude $\forall i, P_i = V_i \setminus \{i\}$.

Symétrie $\forall i, j (j \in P_i), key_i(j) = key_j(i)$.

Confidentialité $\forall i, j (j \in P_i), \nexists k$ tel que $(V_i \cap V_j) \subset V_k$.

Il est facile d'observer que l'algorithme proposé bénéficie des 2 dernières propriétés, mais pas de la première. En effet, la première propriété dépend de la position géographique des capteurs : c'est un paramètre du problème, et toutes les configuration ne permettent pas de remplir la condition *Cond*. Cependant, nous verrons dans la suite que les simulations réalisées montrent qu'une bonne partie des capteurs voisins remplissent la condition.

L'exploitation cryptographique des clés sort du contexte de ce document. Néanmoins, une exploitation simple de la clé consiste, pour crypter, à faire un `xor` (ou exclusif) entre le message à envoyer et la clé. Décrypter le message à destination revient alors à faire la même opération. Soit $cle_i(j)$ et $cle_j(i)$ les clés obtenues respectivement par i et j, et soit m_{ij} et c_{ij} le message clair et chiffré transmis de i à j. Voici une proposition d'exploitation des clés :

$$i \textit{ encrypte le message} : \quad c_{ij} = m_{ij} \oplus key_i(j) \tag{4.1}$$

$$j \textit{ décrypte le message} : \quad m_{ij} = c_{ij} \oplus key_j(i) \tag{4.2}$$

Une dernière observation concernant la cryptographie : on peut observer que chaque noeud de CN_{ij} peut entendre tout cryptogramme émis par i ou par j. Ceci a un intérêt : chaque processus k de CN_{ij} peut compter le nombre de bits cryptés émis avec sa clé bk_k et émettre une nouvelle clé bk'_k lorsque la première a été trop utilisée. Il est même possible d'émettre un nouveau bit de clé à chaque bit utilisé, assurant des mots de passe à usage unique, inviolables. Évidemment, ceci est plus coûteux, mais aussi plus sûr : dans ce sens, l'algorithme permet à l'administrateur de choisir le compromis efficacité-sécurité qu'il désire.

Un noeud byzantin n'a que peu d'intérêt à tricher ici : grâce à la symétrie des relations il est peu possible de mentir sur son voisinage sous peine d'être détecté. Supposons qu'un conflit éclate entre un byzantin b et un noeud i. Les noeuds de CN_{bi} vont détecter une incohérence dans la déclaration de voisinage des deux noeuds. Que b mente en ajoutant ou en retirant i de son voisinage, la procédure est la même : le noeud qui prétend avoir l'autre comme voisin doit pour prouver ses dires émettre la clé émise par l'autre noeud. S'il n'en est pas capable c'est qu'il ment. Par exemple, si b retire i de ses voisins, i apporte la preuve de son innocence en exhibant bk_b. Enfin, b peut s'ajouter des voisins fictifs mais c'est sans effet puisque ces voisins ne seront dans aucun voisinage commun.

De la sécurité du voisinage à la sécurité à plusieurs sauts

Nous avons maintenant la possibilité d'établir des liens sécurisés (cryptés) entre voisins. L'objectif de cette deuxième partie est de transformer ces secrets de voisinage en capacité pour les noeuds de communiquer de façon confidentielle avec presque tous les noeuds du système. Nous allons d'abord étendre ces liens de voisinage en liens sécurisés à deux sauts, puis exploiter le maillage ainsi créé pour faire transiter les messages cryptés.

D'un saut à deux sauts : utilisation de chemins disjoints Nous utilisons ici une technique traditionnelle d'établissement de clés pour créer des liens sécurisés à deux sauts. Pour cela, nous introduisons la notion de *chemin 1-protégé*. Soit $(i, j) \in E$ une arête de G. Cette arête est protégée si *Cond*$_{ij}$ est vraie (les deux extrémités de l'arête partagent une clé). Un chemin dans G est un ensemble d'arêtes, c'est un chemin 1-protégé ssi toutes ses arêtes sont protégées. Soit p^1_{ij} un chemin 1-protégé reliant i à j. On note $\mathcal{C}p^1_{ij}$ les noeuds composant ce chemin.

Il est possible d'établir un lien sécurisé à 2 sauts entre deux noeuds i et j ssi :

$$\textit{Cond}^2_{ij} : \exists \textit{ deux chemins 1-protégés } p^1_{ij}, p'^1_{ij} \textit{ tels que}$$
$$\mathcal{C}p^1_{ij} \cap \mathcal{C}p'^1_{ij} = \{i, j\} \wedge |\mathcal{C}p^1_{ij}| \geq 2 \wedge |\mathcal{C}p'^1_{ij}| = 2$$

Si $\textit{Cond}^2_{ij}$ est vérifiée, i génère deux clés bk_i et bk'_i, et les envoie respectivement à travers p^1_{ij} et p'^1_{ij}. Comme les chemins sont disjoints, seuls i et j peuvent connaître la totalité de la clé. A titre d'exemple, la figure 4.4 illustre l'établissement d'une clé secrète à deux sauts entre les noeuds A et B.

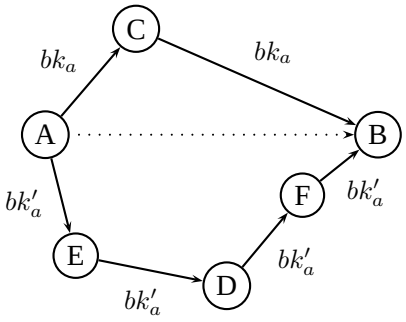


FIG. 4.4 – Exemple d'utilisation des chemins disjoints : une partie de la clé transite par C, l'autre partie transite par E, D et F

De deux sauts au multisaut, ou comment créer des chemins disjoints dans le même chemin. Nous allons maintenant décrire la mécanique de transmission d'un message crypté entre deux noeuds distants de plusieurs sauts. L'intuition de base est liée à l'observation suivante : soit un chemin $p^1_{n_1 n_l}$ reliant deux processus n_1 et n_l. Supposons que le chemin est de la forme $\{n_1, n_2, \ldots n_l\}$. Si chaque noeud n_k du chemin a un lien 2-protégé au noeud $n_k + 2$ (*i.e.* On a $\forall k \in < 1 \ldots l-1 >, \textit{Cond}^2_{n_k n_{k+2}}$) alors le chemin $p^1_{n_1 n_l}$ définit deux chemins disjoints (on suppose l impair) : le chemin $\{n_1, n_3, n_5, \ldots, n_l\}$ et le chemin $\{n_1, n_2, n_4 \ldots, n_{l-1}, n_l\}$. La figure 4.5 illustre ces deux chemins, ainsi que leurs clés respectives : l'un est en pointillés, l'autre est en trait discontinu.

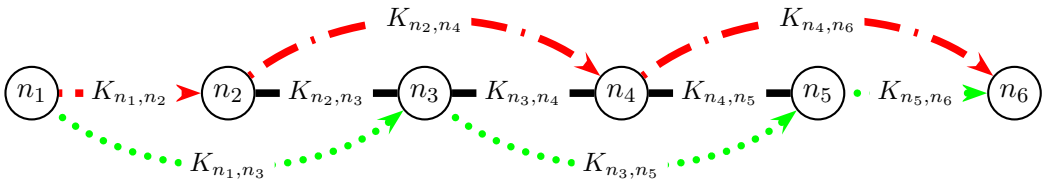


FIG. 4.5 – Exemple de chemin $\mathcal{C}p^2_{ij}$

Nous appelons ce chemin un $p^+_{n_1 n_l}$. De façon plus formelle, pour qu'un chemin p_{ij} soit un chemin p^+_{ij}, il doit remplir les conditions suivantes :

- p_{ij} est un p^1_{ij}
- $\forall \{A, B, C\}$ sous chemin de p_{ij} on a : $\textit{Cond}^2_{n_k n_{k+2}}$

Maintenant, supposons que i veuille envoyer un message M à j et qu'il existe un p_{ij}^{+}. Soit $\{i, n_1, \ldots, n_k, j\}$ ce chemin. Le noeud i divise M en deux parties telles que $M = M_a + M_b$ (quelque soit le sens de l'opération $'+'$, du moment qu'elle est déterministe et commutative). Le noeud i encrypte M_a pour n_2 (avec la clé qu'il partage avec ce noeud), et encrypte M_b pour n_1, puis envoie un seul message à n_1 contenant M_a, M_b et l'identité de n_2. Le noeud n_1 décrypte la partie du message qu'il peut décrypter (ici M_a), encrypte M_a pour n_3, puis envoie à n_2 : M_a, M_b et l'identité de n_3. Ainsi de suite.

Ce mécanisme est illustré par la figure 4.6 : le message est transmis selon les pointillés, mais les arcs indiquent qui crypte quelle partie pour qui (*i.e.* deux extrémités d'une flèche décryptent la même partie du message). Il est important d'observer que pour échanger des messages de cette façon, i et j n'ont pas besoin d'échange préalable : chaque noeud n_i routant le message de manière gloutonne choisit le noeud n_{i+2} le plus proche de la destination.

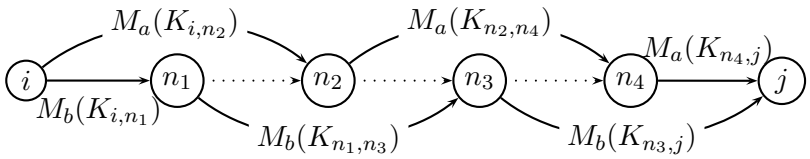


FIG. 4.6 – Exemple de transmission multisaut

4.2.5 Simulations

La possibilité d'établissement d'un lien sécurisé, et plus généralement d'une route sécurisée entre deux noeuds i et j dépend fortement de la disposition des noeuds dans le réseau, autrement dit de la structure du réseau. La question qui se pose maintenant est « à quelle fréquence la structure du réseau permet l'établissement de liens sécurisés ? ». Pour cela, nous avons réalisé des simulations à l'aide d'un simulateur de réseau de capteurs. Nous simulons 500 capteurs répartis uniformément sur un plan de 500×500 unités de distance. En modifiant la portée des capteurs, nous modifions la densité du réseau.

Point de vue local

La condition première pour l'établissement de routes sécurisées est la condition *Cond*. Le paramètre influant sur la fréquence d'occurrence de bonnes configurations est la densité du système : plus un noeud a de voisins, plus il a de chances d'avoir des liens protégés.

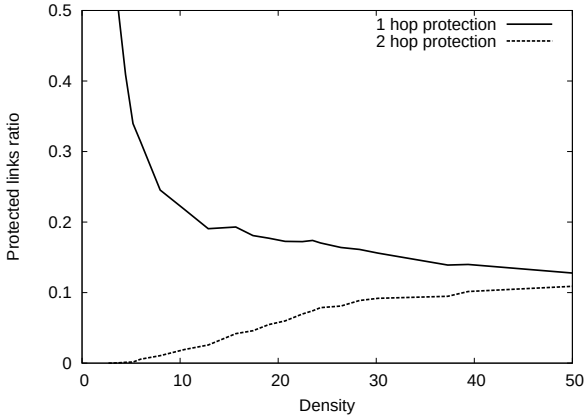
En premier, nous observons le nombre de canaux 1-protégés NL_s dans le système S tel que :

$$NL_s = |\{\mathit{COND}_{ij} | \forall i \in S, \forall j \in V_i\}|$$

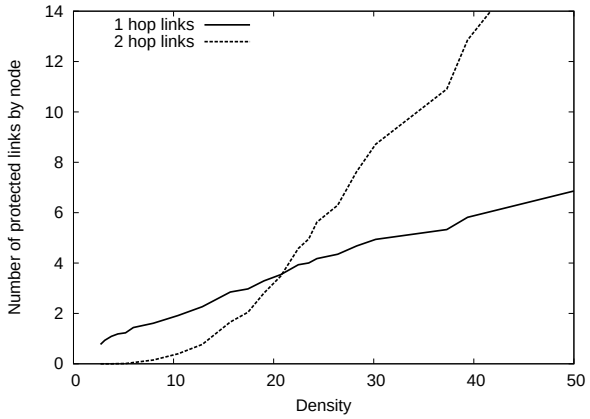
Bien sûr, cet indicateur doit être rapporté à la quantité de liens du graphe G.

$$rl_s = \frac{NL_s}{2|E|}$$

Nous utiliserons similairement les mesures NL_S^2 et rl_S^2 pour étudier les taux d'apparition des liens à 2 sauts (c'est-à-dire la fréquence de satisfaction de la condition Cond^2).



(a)



(b)

FIG. 4.7 – Impact de la densité sur le nombre de liens du système

La figure 4.7(a) représente rl_s et rl_s^2 en fonction de la densité du système. La figure 4.7(b) représente elle NL_S et NL_S^2 toujours en fonction de la densité. Ainsi, on peut y lire que dans un système ayant une densité de 40 noeuds par portée de communication, un noeud peut espérer avoir des liens sécurisés avec 14% de ses voisins (figure de gauche) ce qui donne entre 5 et 6 liens par noeud (figure de droite). Ces figures illustrent le fort impact de la densité : une forte densité réduit la probabilité pour qu'un lien donné soit sécurisé, mais vu qu'une grande densité implique un grand nombre de liens potentiellement sécurisés, le nombre de liens sécurisés par noeud augmente avec la densité.

Un constat s'impose : on est loin de la propriété de complétude décrite section 4.2.3. Néanmoins, si chaque noeud a 5 ou 6 voisins avec qui il peut établir des communications sécurisées cela peut s'avérer suffisant pour établir des communications sécurisées avec la majorité des noeuds du système. Pour répondre à cette interrogation, il faut analyser avec un point de vue global la structure de liens sécurisés que nous créons ici.

Point de vue global

Pour garantir la sécurité, il faut que les noeuds puissent transmettre de l'information à travers le réseau en utilisant seulement des liens protégés. La question que nous posons maintenant est « quel est la proportion de noeuds pouvant communiquer de façon sécurisée ? ». Soit $G(V, E)$ le graphe du système comme introduit dans le modèle. On a $(i, j) \in E \Leftrightarrow$ *i et j sont voisins*. Soit maintenant $G_1(V, E_1)$ et $G_2(V, E_2)$ deux graphes tels que $(i, j) \in E_1 \Leftrightarrow$ *Cond*$_{ij}$ et $(i, j) \in E_2 \Leftrightarrow$ *Cond*$^2_{ij}$. Soit maintenant S_1 et S_2 les plus grands ensembles de sommets connexes des graphes G_1 et G_2 respectivement.

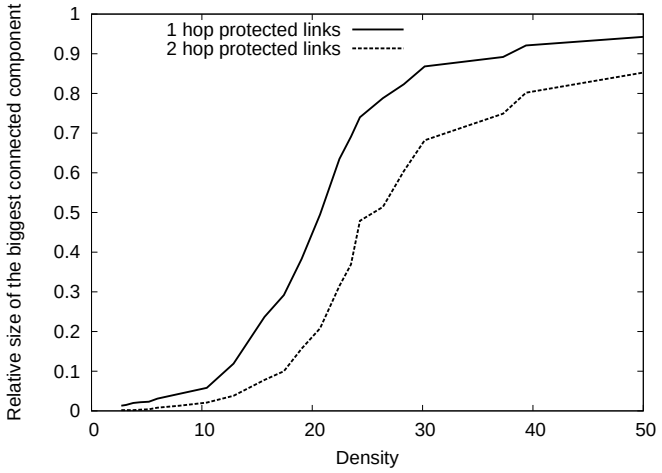
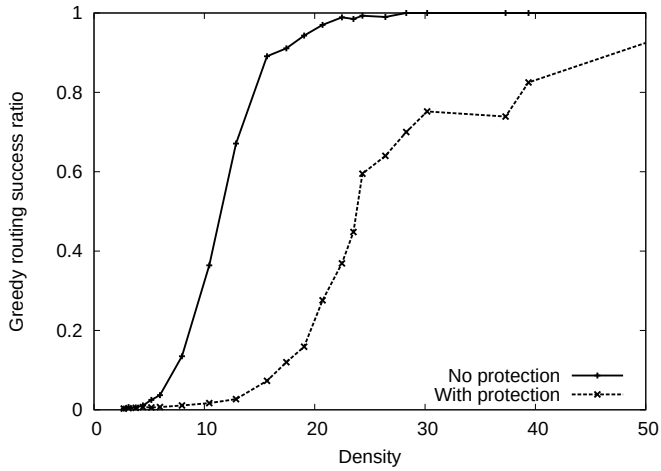


FIG. 4.8 – Impact de la densité sur le nombre de noeuds partageant des clés

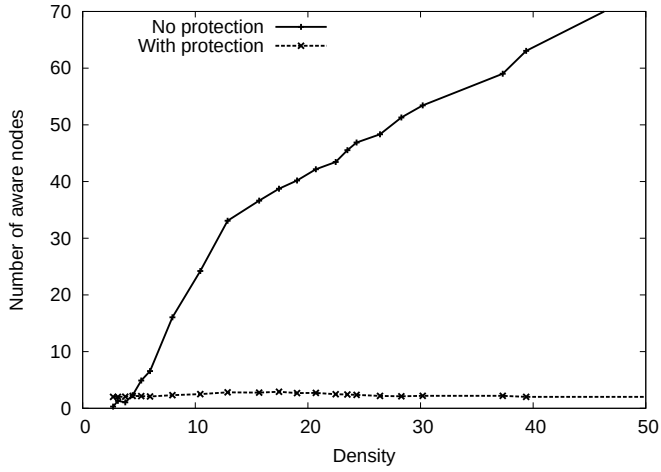
La figure 4.8 représente les variations de $|S_1|/|E|$ (courbe pleine) et $|S_2|/|E|$ (courbe pointillée). Les deux courbes présentent un fort effet de percolation qui illustre l'effet de la densité sur la connectivité globale. De plus, ces courbes montrent que l'algorithme permet à une grande partie des noeuds de communiquer si la densité du système le permet.

Gain de confidentialité

La figure 4.9 présente d'intéressants résultats concernant le routage à l'aide de liens protégés. L'expérience est la suivante : chaque noeud du système découvre ses liens protégés et essaie d'en-



(a)



(b)

FIG. 4.9 – Impact du protocole sur le routage glouton et la confidentialité

voyer un message au puits (qui est choisi arbitrairement comme le noeud le plus proche du centre du système). L'envoi se fait dans un cas en utilisant aucune protection et dans l'autre cas en utilisant les liens protégés établis par le protocole. Chaque point de la courbe est la moyenne de 20 expériences indépendantes. La figure 4.9(a) représente le taux de succès d'un routage glouton sur les coordonnées physiques des noeuds (on suppose ici, pour les besoins du routage, qu'ils y ont accès). On peut y observer que le routage glouton utilisant des liens protégés est efficace dès que la densité est assez forte.

La figure 4.9(b) compare le nombre moyen de noeuds pouvant écouter un message échangé entre un noeud et le puits. Si le noeud source est directement relié par un chemin p^+ au puits, seuls ces deux noeuds peuvent comprendre le message. Sinon, chaque noeud relaie le message à un noeud plus proche du puits, jusqu'à pouvoir emprunter un chemin p^+ le rapprochant et ainsi de suite. Rappelons qu'il n'est effectué aucune découverte de chemins a priori : tous les routages sont gloutons et un noeud décide en connaissant seulement ses voisins et les positions de ses voisins à deux sauts. Il est intéressant d'observer que dans la plupart des cas, le noeud source et le puits sont directement connectés par un chemin p^+, ils peuvent donc communiquer confidentiellement. Sans protection, le nombre de noeuds ayant accès au contenu du message augmente rapidement avec la densité, illustrant le danger de la combinaison des ondes radio et des communications multi-sauts dans en réseau non sécurisé.

4.2.6 Structures régulières

Dans les sections précédentes, nous avons étudié le cas de capteurs disséminés aléatoirement dans un plan. Nous avons vu que la quantité de liens sécurisés était bonne mais pas parfaite : tous les noeuds n'ont pas de liens sécurisés. Ces noeuds n'ont pas de liens sécurisés parce que la structure locale de leur voisinage ne le permet pas. Inversement, nous nous intéressons ici aux structures permettant l'apparition d'un grand nombre de ces liens. Il existe même des structures où tous les liens sont sécurisés autrement dit des graphes $G(V, E)$ tels que $\forall (i, j) \in E, \mathit{Cond}_{ij}$.

Bien sûr, utiliser de telles structures suppose l'intervention d'un humain pour placer chaque noeud à la bonne place. La figure 4.10 présente deux exemples de telles structures avec trois portées différentes. Rappelons que deux paramètres définissent le graphe de communication : le placement des noeuds et leur portée radio. Il est possible d'observer sur ces exemples que quelque soit la portée des noeuds, les deux noeuds noirs peuvent communiquer de façon confidentielle : l'intersection des voisinages de leurs voisins communs (en rectangle) ne contient que les noeuds noirs.

Caractérisation des structures favorables à l'établissement des liens sécurisés

Nous allons ici proposer une caractérisation simple des structures permettant l'établissement de routes sécurisées entre toute paire de noeuds du système. Pour cela nous introduisons la notion de différentiabilité locale et l'utilisons pour caractériser les structures solution. Soit un graphe $G(V, E)$, pour tout noeud $i \in V$ nous appelons V_i l'ensemble des voisins de i, *i.e.* $j \in V_i \Leftrightarrow (i, j) \in E$. Nous supposons le graphe symétrique : $i \in V_j \Leftrightarrow j \in V_i$.

Definition 4 *G est localement différentiable si*

$$\forall x \in V, \forall y, z \in V_x^2, (V_x \cap V_y) \neq (V_x \cap V_z)$$

L'idée derrière la notion de différentiabilité est que du point de vue d'un noeud (point de vue local), tous les voisins diffèrent par leur voisinage. Soit $K_{xy} = \bigcap_{k \in V_x \cap V_y} V_k$. Rappelons que nous avons un lien sécurisé entre x et y ssi $K_{xy} = \{x, y\}$ (c'est la condition Cond_{xy}).

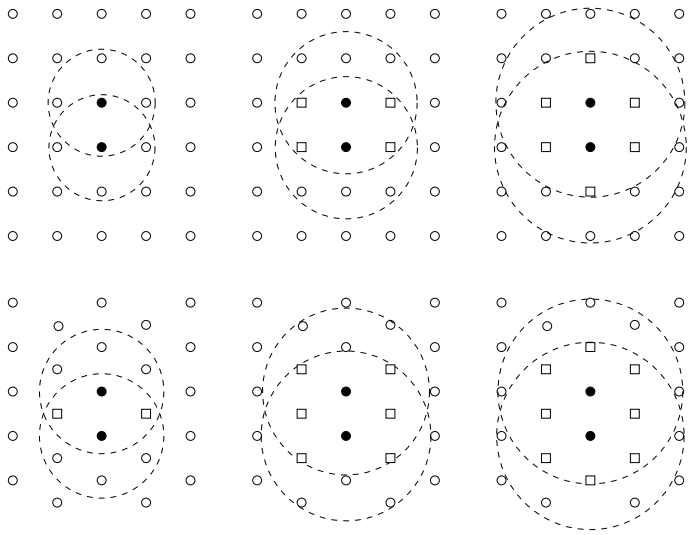


FIG. 4.10 – Structures particulières

Théorème 12 *Soit G un graphe localement différentiable. Alors pour tout noeud $x \in V$, pour tout $y \in V_x$, il existe une route sécurisée (*i.e. *un chemin p^1_{xy}) entre x et y.*

Preuve La démonstration se fait par récurrence sur la proposition : soient deux noeuds x et y voisins dans G tels que $|K_{xy}| = n$ alors il existe un chemin p^1_{xy}.

Nous utilisons une récurrence sur la taille de l'intersection des voisinages des voisins communs. Observons d'abord que $\{x, y\} \in K_{xy}$ puisque $\{x, y\} \in V_x$ et$\{x, y\} \in V_y$ et que x et y sont voisins. Donc si $|K_{xy}| = 2$, nécessairement $K_{xy} = \{x, y\}$: x et y partagent un lien sécurisé (une route d'un seul saut) ce qui prouve la proposition pour $n = 2$.

Supposons maintenant la proposition vraie pour $k < n$, avec $n > 2$. Soit x, y deux noeuds voisins tels que $|K_{xy}| = n$. On a $n \geq 3$ donc il existe a tel que $a \neq x, a \neq y$ et $a \in K_{xy}$.

$$
\begin{aligned}
a \in K_{xy} &\Leftrightarrow \forall \ell \in V_x \cap V_y, a \in V_\ell \\
&\Leftrightarrow \forall \ell \in V_x \cap V_y, l \in V_a \\
&\Leftrightarrow V_a \supset V_x \cap V_y \\
&\Rightarrow V_x \cap V_a \supset V_x \cap V_y \textit{ et } V_y \cap V_a \supset V_x \cap V_y \\
&\Rightarrow \cap_{k \in V_x \cap V_a} V_k \subset \cap_{k \in V_x \cap V_y} V_k \textit{ et } \cap_{k \in V_y \cap V_a} V_k \subset \cap_{k \in V_x \cap V_y} V_k \\
&\Rightarrow K_{xa} \subset K_{xy} \textit{ et } K_{ya} \subset K_{xy}
\end{aligned}
$$

G est localement différentiable et puisque$V_x \cap V_a \supset V_x \cap V_y$ *et* $V_y \cap V_a \supset V_x \cap V_y$, il existe un noeud z tel que $z \in V_x \cap V_a, z \notin V_x \cap V_y$ et symétriquement, il existe z' tel que $z' \in V_y \cap V_a, z' \notin V_x \cap V_y$. Or $z \notin V_y \Leftrightarrow y \notin V_z \Rightarrow y \notin K_{xa}$. Similairement en utilisant z' on déduit $x \notin K_{ya}$.

D'où

$$K_{xa} \subsetneq K_{xy} \textit{ et } K_{ya} \subsetneq K_{xy}$$

Donc $|K_{xa}| < |K_{xy}| = n$ (idem pour K_{ya}). D'après l'hypothèse de récurrence, il y a donc un chemin sécurisé entre x et a et entre a et y. On pose alors $p^1_{xy} = p^1_{xa} + p^1_{ay}$, ce qui clos la récurrence. $\square_{\textit{Théorème}}$

Corollaire 4 *Soit un graphe G connexe et localement différentiable. Alors* $\forall x, y \in V^2, \exists p^1_{xy}$

Preuve

La preuve découle du théorème 12 et du fait que G est connexe. En effet puisque G est connexe, il existe un chemin p_{xy} reliant x à y. Soit $\{n_1, \ldots, n_k\}$ ce chemin (on pose par commodité $x = n_1$ et $y = n_k$). Puisque G est localement différentiable, on a $\forall i \in < 1..n-1 >$ un chemin sécurisé p^1_{ii+1}. On pose alors $p^1_{xy} = \sum_{i=1}^{n-1} p^1_{ii+1}$. $\square_{Corollaire}$

4.2.7 Comportement byzantin

Dans le modèle présenté précédemment, nous étudions un modèle particulier de noeud byzantin qui pourrait être décrit comme le « noeud byzantin participant ». Dans ce modèle, il faut que les noeuds byzantins exécutent une partie de l'algorithme comme les autres noeuds. Ce modèle est réaliste dans le contexte d'une application fournissant certains services auxquels les noeuds byzantins souhaitent accéder. A cette fin, ils doivent se faire passer pour des noeuds corrects. Si l'on reprend l'image du poker, nous nous intéressions à empêcher les joueurs de tricher, sans chercher à contraindre le comportement du spectateur. Puisque le spectateur ne joue pas, il ne peut rien gagner dans tous les cas, il peut donc « tricher » en regardant les mains des autres (on ne traite pas la collusion, particulièrement celle entre un spectateur et un joueur).

Néanmoins, il est intéressant de voir l'efficacité du protocole face à des noeuds byzantins *muets* : dans ce modèle enrichi, les noeuds byzantins peuvent aussi ne rien émettre du tout et écouter simplement les messages, compromettant éventuellement quelques clés.

Pour évaluer la menace, nous avons conduit des expériences. En utilisant le même protocole expérimental nous avons placé au hasard des noeuds byzantins muets dans le réseau. Puis pour chaque byzantin, nous avons calculé s'il était capable de compromettre un chemin p^+. Ceci peut être vu comme une utilisation de la méthode de Monte-Carlo pour évaluer la surface vulnérable du réseau, la surface vulnérable étant la zone où un noeud byzantin muet peut intercepter une communication. Notons que pour intercepter une communication p^+, un byzantin doit intercepter deux liens 2-protégés.

La figure 4.12 représente la probabilité pour un noeud byzantin muet de tomber dans une zone vulnérable du réseau. Nous considérons deux cas : la courbe noire représente la probabilité pour ce type de noeud d'intercepter deux liens 2-protégés quelconques. La courbe grise représente la probabilité d'intercepter des liens consécutifs utilisables comme route menant au puits. Le puits est toujours arbitrairement choisi comme le noeud le plus au centre du système. Cette courbe montre qu'un noeud byzantin placé aléatoirement dans un système avec une densité de voisinage de 35 à 20% de chance d'intercepter deux liens 2-protégés quelconques, mais seulement 3% de chances d'intercepter un message allant au puits.

Cette différence s'explique géométriquement : rappelons qu'il faut intercepter deux liens 1-protégés pour intercepter un lien 2-protégé. Il faut donc théoriquement intercepter 4 liens 1-protégés pour intercepter un chemin p^+. Cependant, lorsque l'angle formé par deux liens 2-protégés est petit, ce sont souvent les mêmes noeuds qui fournissent les clés des deux liens, et il arrive que le même lien 1-protégé soit utilisé pour établir les deux liens 2-protégés : l'interception est plus facile. La figure 4.11 schématise ce phénomène : la zone de vulnérabilité (en hachures) est plus grande pour le chemin A, B, C que pour le chemin D, E, F. Néanmoins, il est peu probable qu'un message emprunte le chemin A, B, C puisque cela constitue presque un demi-tour.

FIG. 4.11 – Illustration de l'impact de l'angle des routes sur la zone de vulnérabilité : plus l'angle est petit plus la zone vulnérable est grande.

4.3 Conclusion

A nouveau, nous avons vu deux études dans ce chapitre. La première étude a été publiée dans sa version originale [HMT07].

La première étude traite du problème du consensus dans un environnement partiellement synchrone face à un adversaire byzantin. Cette partie illustre ainsi l'emploi de contraintes sur la structure de communication (l'ajout de liens synchrones) pour résoudre des problèmes impossibles sinon (le consensus asynchrone). Cette partie apporte les contributions suivantes :

- Nous présentons, dans un modèle avec authentification où les hypothèses de synchronie sont particulièrement faibles, un algorithme réalisant le consensus byzantin. Ce protocole est prouvé.
- Ce protocole est très simple en comparaison aux autres protocoles à la Paxos.
- Dans les bonnes configurations (le premier coordinateur est une $2t$-bisource), le protocole termine en 5 étapes de communication, ce qui est une propriété très utile : dans les configurations normales c'est un cas fréquent.

La deuxième étude traite de l'établissement de secrets partagés dans un réseau de capteurs fixes. Pour cela, nous exploitons les propriétés locales de la structure (les configurations du voisinage). L'approche ici est toujours d'exploiter la structure, mais d'une manière différente : nous ne contraignons plus la structure par le modèle, mais nous conditionnons le résultat à la vérification de certaines propriétés (locales) par la structure. Il est ainsi possible que la solution proposée soit applicable seulement pour une partie des entités d'une structure. Cette partie apporte les contributions suivantes :

- Nous proposons d'abord un protocole très simple, permettant d'établir une clé secrète partagée par une paire de noeuds voisins dans le réseau de capteurs. Pour cela, nous supposons les communications symétriques. La capacité de deux noeuds à partager un secret dépend uniquement de la configuration locale de leur voisinage.
  Aucune préconfiguration, ni matériel spécifique n'est nécessaire, ce qui est une différence majeure par rapport aux mécanismes d'établissement de clés existants. Dans cette idée, nous proposons une solution purement algorithmique déployable sur des capteurs identiques et génériques. Cette solution utilise ainsi la position aléatoire du capteur comme source d'asymétrie pour créer des secrets partagés.

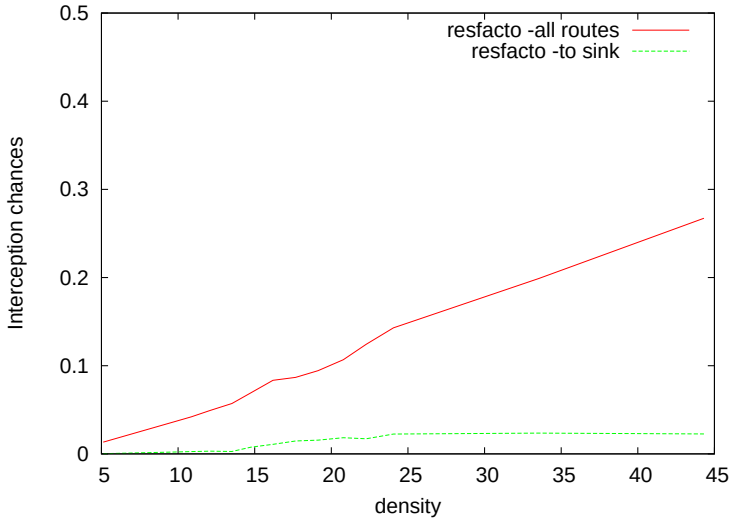


FIG. 4.12 – Probabilité d'interception d'un chemin p^+ par un noeud byzantin muet

– Nous proposons ensuite un autre protocole permettant à n'importe quelle paire de capteurs d'établir une clé secrète partagée. Cet algorithme repose sur l'établissement des clés locales effectuées par le premier protocole.
  Nous montrons que sur des réseaux où les noeuds sont aléatoirement répartis, les propriétés nécessaires à l'établissement de tels canaux sont souvent respectées pour permettre à la majorité des noeuds de communiquer. Nous prouvons aussi que cette solution est compatible avec un routage glouton.
– Nous introduisons la propriété de différentiabilité locale d'un graphe. Nous montrons ensuite qu'il existe des chemins protégés entre tous les noeuds d'un graphe localement différentiable.

Chapitre 5

Expliciter les structures

Dans le chapitre précédent, nous avons vu comment exploiter les structures naturelles des systèmes répartis afin de construire des solutions efficaces et robustes. Si l'on observe la façon dont les solutions ont été établies dans le chapitre précédent, il est possible de faire plusieurs remarques :

- Les algorithmes présentés agissent comme filtres :
    1. ils isolent et travaillent des sous-parties de la structure : l'étoile des liens partant d'un coordinateur pour le consensus byzantin, un lien entre deux noeuds pour les réseaux de capteurs.
    2. ils parcourent (souvent exhaustivement) les sous-parties possibles de la structure (chaque coordinateur à des instants différents, chaque noeud et son voisinage).
    3. ils testent et discriminent ces sous-parties selon qu'elles conviennent ou pas.
- Lorsque la sous-structure isolée ne convient pas, l'objectif est de garantir une cohérence dans le système.
- Lorsque la sous-structure isolée convient, le plus souvent, ils ne font rien (sinon rapidement exploiter cette structure). Le rôle de la preuve de l'algorithme consiste à montrer que la solution du problème et la sous-structure identifiée sont synonymes (l'une s'écrit en termes de contraintes sur les interactions, l'autre en termes de contraintes sur des variables). Identifier la sous-structure est donc souvent déjà obtenir la solution.

Dans ce sens, nous isolons une structure qui fonctionne et bâtissons la solution dessus.

Dans ce chapitre, nous allons aussi utiliser les structures sous-jacentes du réseau de communication, mais cette fois pour transformer celle-ci en une nouvelle structure. Contrairement à la structure de communication, la structure solution aura un caractère explicite et logique. En effet, chaque noeud sera conscient de sa position au sein de celle-ci, elle est explicite. De plus, les noeuds sont maintenant reliés au sein de cette structure par des variables (comme la position), et non pas par des contraintes physique (distance, temps de transfert) : la structure est logique.

Cette position, que l'on peut voir comme une identité au sein de la structure créée, a donc l'intérêt d'être une variable logique et d'être explicite. Cela permet de la manipuler et de l'exploiter facilement. La position a un autre intérêt : elle est cohérente dans la structure. Cela veut dire que si l'on connaît l'identité d'un noeud au sein de cette structure, on peut déduire beaucoup d'informations par rapport à la relation physique capturée par la construction de la structure logique.

C'est ici que la construction de structures logiques va trouver son importance : elle va permettre de stocker de l'information sur les relations entre entités. Cette information est stockée afin d'être utilisable facilement par tous les noeuds du réseau, et ainsi plutôt que de recalculer à chaque fois chaque information, les noeuds se réfèrent ensuite à la structure, amortissant le coût de sa construction par son utilisation régulière. A ce titre, il est important de construire les structures

logiques sur des relations durables à travers le temps, puisque l'idée est de conserver longtemps la structure pour rentabiliser l'investissement de sa création. Si la structure physique évolue vite, il n'est pas rentable de construire une structure logique qui deviendra caduque aussi rapidement.

Nous allons dans ce chapitre présenter des travaux portant sur la construction d'une des plus communes structures logiques : un système de coordonnées. Ces travaux se situent à nouveau dans le contexte des réseaux de capteurs. L'objectif va être ici de cartographier les relations de voisinages des capteurs en leur procurant des coordonnées. Ces coordonnées ont une cohérence : par exemple, si l'on appelle $x(i) = \{x_1(i), x_2(i) \dots x_d(i)\}$ les coordonnées du noeud i, pour tout noeud j voisin de i, on a $\forall k \in <1..d> |x_k(i) - x_k(j)| \leq 1$ (il n'existe pas de voisin qui ait une coordonnée différente de plus de 1 saut).

C'est la traduction de cette relation de voisinage en relation logique qui va permettre une manipulation beaucoup plus aisée de la structure de communication physique au niveau algorithmique. Cette traduction va dans notre cas permettre le routage dans le réseau de capteurs, ainsi que la définition de partitions. En résumé, nous allons ici traduire une structure de communication implicite en structure logique explicite et donc exploitable.

5.1 Coordonnées virtuelles pour les réseaux de capteurs

L'auto-structuration définit la capacité d'un système à laisser émerger une structure spécifique sans aide extérieure. Il s'agit d'une capacité vitale des systèmes autonomes, surtout pour permettre à ceux-ci de passer à l'échelle [DWB+07]. Dans les réseaux de capteurs par exemple, l'auto-structuration est importante pour de nombreuses opérations classiques telles l'élection de leader, l'équilibrage de charge, le routage ou encore l'économie d'énergie. Un exemple classique de structuration consiste à partitionner l'espace sur lequel les capteurs sont réunis afin d'agréger les informations d'une partie donnée et ainsi économiser de l'énergie pour la transmission des informations.

La complexité d'un algorithme d'auto-structuration dépend fortement de la quantité de connaissance disponible initialement. Si les noeuds ont une connaissance complète du système, la structuration est triviale. A l'opposée, si chaque noeud ne connaît que son voisinage, s'assurer qu'une structure cohérente à l'échelle du réseau émerge de la somme des décisions individuelles des entités est difficile.

Nous définissons la *connaissance externe* comme la connaissance attribuée aux noeuds par une source extérieure. Celle-ci peut provenir d'un dispositif, un GPS par exemple, ou d'une préconfiguration du capteur, en préconfigurant le nombre d'entités du système par exemple, ou encore d'une hypothèse (la zone de déploiement est rectangulaire), etc. Nous l'opposons à la *connaissance interne* qui consiste en l'information que chaque capteur acquiert par l'observation du réseau et l'échange de messages.

Cette distinction entre les sources de connaissance souligne un dilemme. Structurer un réseau nécessite que chaque noeud ait acquis une certaine quantité de connaissance. Plus une structure repose sur de la connaissance externe, moins cette structure est robuste puisqu'elle repose alors sur des facteurs externes (disponibilité et précision du GPS, régularité de la zone de déploiement) qui, s'ils sont mis en défaut, empêchent la structuration de réussir. De l'autre côté, plus une structure repose sur une connaissance interne, plus le coût de mise en oeuvre de cette structure est élevé : il faut acquérir cette connaissance, et cela engendre des coûts de communication et de calcul. Le dilemme est donc de choisir entre robustesse et économie (on retrouve d'ailleurs un compromis classique de la construction et, plus généralement, de l'ingénierie).

Dans cette section, nous présentons un mécanisme de structuration robuste permettant l'organisation géographique d'un système (ce terme sera défini par la suite). Ce mécanisme de structuration construit une solution à partir de très peu d'informations : la connaissance de chaque noeud

est limitée à la connaissance de son identité. A partir de cette organisation, nous verrons qu'il est possible d'assigner différents rôles aux noeuds selon leur position dans la structure créée. Ces travaux sont dans ce sens particulièrement adaptés aux réseaux de capteurs.

La structuration du réseau se fait ici par la création d'un système de coordonnées virtuelles. Créer un tel système n'est pas trivial lorsque les noeuds ont peu d'information à disposition. Cependant, les avantages d'une telle approche sont multiples :

- la solution étant algorithmique, elle ne nécessite pas de matériel dédié, elle est donc plus économique.
- les coordonnées construites sont virtuelles, ce qui rend le routage plus robuste face aux trous de densité dans le réseau, etc.
- l'obtention ou l'attribution de coordonnées géographiques aux noeuds n'est pas toujours possible. A ce titre la structuration virtuelle fournit une approche complémentaire.

La plupart des approches concernant le positionnement dans les réseaux de capteurs [HWLC01, BOCB07, CCDU05, BW04, Gus03, RRP+03] reposent sur des hypothèses spécifiques (présence de systèmes de mesure du signal ou encore utilisation d'entités bien localisées - les ancres). Plus récemment, des solutions n'utilisant pas de points bien localisés ont été présentées [RRP+03, MOWW04, BPA+06]. En utilisant ces solutions, puisque aucune position n'est disponible, les noeuds se voient attribuer des coordonnées virtuelles.

Dans cette partie, nous introduisons les contributions suivantes :

- Un algorithme simple et entièrement réparti d'attribution de coordonnées virtuelles `Vincos`. Notre algorithme repose sur l'exploitation des noeuds de bordure (*i.e.* les noeuds situés en périmètre du système).
- Une approche de structuration géométrique `NetGeos`. Nous présentons les intérêts d'une telle structuration ainsi que comment l'effectuer au dessus de `Vincos`.

5.1.1 Travaux connexes

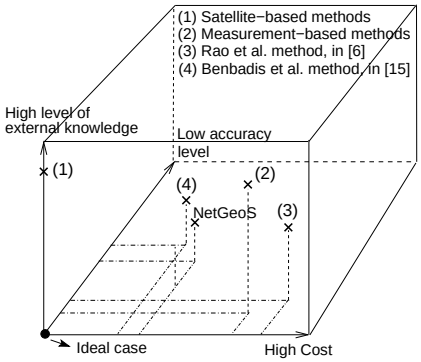


FIG. 5.1 – Vincos en termes de coût, précision et connaissance externe.

La figure 5.1 compare les différents systèmes de coordonnées de la littérature par rapport à `Vincos`. Elle compare leurs différences en termes de coût (nombre de messages), précision des coordonnées, et connaissance externe telle qu'elle est définie plus haut. La précision des coordonnées peut être vue comme la résolution des coordonnées, ou encore comme la quantité de noeuds partageant les mêmes coordonnées. Plus un point est proche de l'origine des axes, plus il représente un compromis efficace entre ces critères.

L'importance des systèmes de coordonnées pour les réseaux de capteurs est démontrée par l'abondante littérature traitant du sujet [BW04, HHB+03, HWLC01, MOWW04, NSB03,

RRP+03, BOCB07, CCDU05, SRZF03, Gus03, DPG01, BPA+06]. Dans la catégorie des systèmes de positionnement absolus, nous avons les systèmes de positionnement satellite comme GPS ou Galileo. Equiper tous les capteurs de récepteurs GPS est la solution simple d'apporter le maximum de connaissance externe aux capteurs. Il en résulte un système de coordonnées très précis (point (1) sur la figure 5.1). Cependant cette méthode présente de nombreux désavantages : elle est onéreuse, inefficace énergétiquement (les récepteurs GPS consomment de l'énergie), et ne peut pas marcher si les capteurs sont déployés hors d'une zone de couverture satellite (déploiements extra-terrestres ou déploiements souterrains).

Ce problème peut être résolu en équipant seulement quelques entités (*i.e.* des ancres) avec un tel récepteur satellite et en laissant les autres entités estimer leur position à partir des informations de connectivité ou de dispositifs de mesure du signal (angle d'arrivée du signal, force du signal, utilisation de signaux de nature différente et mesure des temps de propagation) [BW04, MOWW04, BFAF05, CCDU05, SRZF03, Gus03, DPG01]. Ces approches hybrides sacrifient beaucoup de précision pour un coût toujours élevé (point (2) sur la figure 5.1).

Les solutions basées seulement sur la connectivité [BW04, BFAF05, CCDU05] sont algorithmiquement plus simples. Elles ont néanmoins un coût de communication élevé, et la précision du système de coordonnées résultant dépend fortement du nombre d'ancres et de leur position dans le réseau (si les ancres sont mal placées le positionnement peut être de très mauvaise qualité) (point (4) sur la figure 5.1).

Une approche plus attractive consiste à se passer d'ancres [MOWW04, BOCB07, RRP+03]. Ceci engendre des systèmes à coordonnées virtuelles par opposition aux coordonnées géographiques obtenues dans les autres cas. `Vincos` utilise une procédure d'assignement de coordonnées similaire à celle de [BOCB07, RRP+03] du point de vue de la connaissance initiale apportée aux noeuds (presque aucune connaissance). Ces approches considèrent que certains noeuds particuliers sont identifiés (noeuds de périmètre ou noeud central). En particulier par rapport à [RRP+03], `Vincos` propose une exploitation radicalement différente de la connaissance des noeuds constituant le périmètre du système (noeuds de bordure). Il est possible de voir `Vincos` comme la recherche d'ancres optimalement placées pour [BOCB07].

5.1.2 Principe des coordonnées

Le positionnement est une information clé dans la construction et le maintien de systèmes autonomes. Un système de coordonnées procure à chaque noeud une *position* qui est à la fois localement et globalement cohérente. Dans `Vincos`, chaque noeud est configuré initialement avec un paramètre d : la dimension du système de coordonnées. Ainsi, les coordonnées virtuelles d'un noeud i sont représentées sous la forme d'un tuple $(x_1, \ldots, x_d)$, où x_j est la projection du noeud i sur le $j$ème axe de l'espace virtuel à d dimensions. L'unité de distance utilisée est le saut.

Cet espace virtuel est défini comme suit. La bordure de la zone géographique couverte par les noeuds est partitionnée en d "segments". Ces *segments de bordure* peuvent avoir soit la même taille, soit des tailles différentes[1]. Soit un axe j, $1 \leq j \leq d$, du système de coordonnées. La coordonnée x_j est alors la distance, en sauts, du plus court chemin du noeud i au segment j (*i.e.* au noeud le plus proche appartenant au segment). La figure 5.2 illustre ceci : l'espace de coordonnées y est de dimension $d = 3$. Les coordonnées virtuelles de trois noeuds y sont portées : les coordonnées $(2, 4, 2)$ signifient que le noeud est à distance 2 des bordures 1 et 3 et à distance 4 de la bordure 2. Ce système ne nécessite ainsi pas d'ancre ou de noeud particulier.

Les coordonnées de `Vincos` reposent sur l'identification correcte des noeuds de bordure. Le principe est simple : une fois ces noeuds découverts, l'algorithme procède à une élection de leader

[1]Pour des raisons de simplicité, nous présenterons le cas où les segments sont de taille égale. La démarche est identique si les segments ont des tailles différentes.

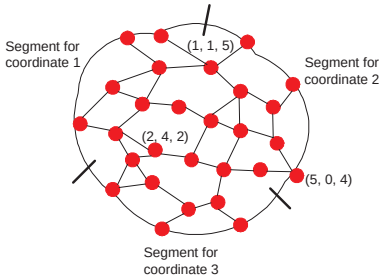


FIG. 5.2 – Un exemple de coordonnées virtuelles.

sur la bordure. Le leader de bordure ainsi élu partitionne ensuite la bordure en d segments, chaque segment représentant un axe du système de coordonnées. Les noeuds de la bordure, lorsqu'ils apprennent à quels axes ils appartiennent, inondent ensuite le réseau de façon à apprendre à chaque noeud du système sa distance à l'axe.

Du point de vue structure, la structuration effectuée par `Vincos` a lieu en deux temps : d'abord, la bordure est structurée, ensuite la structure de la bordure est utilisée pour structurer le reste du système (l'intérieur). Cette approche a plusieurs avantages :

- Comme nous le verrons plus loin, nous définissons la bordure comme un circuit de noeuds. Cette forme de circuit rend sa structuration simple : il n'y a qu'une dimension à configurer. De plus, comme c'est un circuit, un noeud voyant son message faire un tour complet est sûr que tous les noeuds connaissent ce message.
- Les densités autour de la bordure sont plus faibles (les noeuds en périphérie ont moins de voisins). Lorsqu'un noeud émet un message, de l'énergie est dépensée par chaque voisin du noeud émetteur pour recevoir ce message. De plus la probabilité que les communications de deux voisins interfèrent est proportionnelle à la densité. Dans ce sens travailler avec les noeuds de bordure est plus économique.
- Si l'on élimine les cas de réseaux très particuliers, travailler avec la bordure est plus sûr du point de vue de la connexité du système. Lorsqu'un capteur n'a plus d'énergie, il meurt. Si des noeuds aléatoires échangent des messages dans un réseau, ce seront les noeuds centraux, statistiquement plus sollicités, qui seront les premiers à court d'énergie. Ceci engendre un trou dans le centre du réseau qui impacte fortement les performance du système tout entier. A l'opposée fatiguer les noeuds de bordure est moins grave, puisqu'au pire la surface de couverture du réseau est réduite, sans impacter les capacités de communication des autres noeuds.

5.1.3 Définition de la bordure du système

Nous avons vu dans le paragraphe précédent que le système de coordonnées `Vincos` repose sur l'utilisation de la bordure du système. Nous allons ici définir formellement ce qu'est une bordure et passer en revue quelques méthodes de détection de celle-ci.

La notion de bordure

Le modèle du système est défini dans la section 5.1.4. Nous le modélisons ici par un graphe UDG [CCJ90] $G = (V, E \subset (V \times V))$, où un sommet v_i représente le noeud i, et un arc e_{ij} existe si, et seulement si, les noeuds i et j sont voisins.

Un chemin est une suite de sommets telle que chaque sommet est voisin du sommet suivant, et un circuit est un chemin dont le premier et le dernier noeud sont identiques. Un circuit simple est

un chemin dans lequel (hormis le noeud source) chaque noeud apparaît au maximum une fois.

Un tel circuit $c = \{s, c_1, \ldots, s\}$ partitionne V en 3 parties :

– Les sommets appartenant au circuit,
– Les sommets d'un côté du circuit,
– Les sommets d'un autre côté du circuit.

Dans le modèle UDG, on suppose les noeuds positionnés sur un plan à deux dimensions. Soit $p(v_i) = \{x_i, y_i\}$ les coordonnées du noeud i. Le circuit simple c défini alors un polygone $p(c) = \{p(s), p(c_1), \ldots, p(s)\}$. Soit $S(c)$ l'aire du polygone définit par le circuit c. Nous définissons la meilleure bordure b comme le circuit dont le polygone associé a la plus grande aire. Soit C l'ensemble des circuits simples possibles dans G. Le circuit $b \in C$ est une bordure ssi

$$S(b) = \max_{c \in C}(S(c))$$

Il est intéressant d'observer qu'aucune garantie n'existe sur le nombre de noeuds situés à l'extérieur de l'aire du polygone défini par le chemin. En effet, puisque nous ne considérons que les circuits simples, nous calculons ici la bordure du sous-graphe 1-connexe de G. Si l'on prend l'exemple de noeuds disposés en croix (*i.e.* suivant l'axe des abscisses et l'axe des ordonnées d'un repère orthonormé), il est possible d'avoir un graphe de taille arbitrairement grande avec une bordure nulle !

Propriétés du détecteur de bordure

Comme indiqué précédemment, la détection d'une bordure dans les réseaux de capteurs est une tâche difficile si les noeuds ne connaissent pas leur position. Nous allons ici décrire les propriétés que nous attendons d'un détecteur de bordure.

Nous définissons un *noeud de bordure fort* comme un noeud appartenant au circuit définissant la bordure. Nous définissons un *noeud de bordure faible* comme un noeud étant soit noeud de bordure fort, soit voisin d'un noeud de bordure fort.

Un détecteur de bordure est défini ici comme une boite noire, à la manière d'un oracle. Celui-ci procure à chaque noeud i un booléen on_border_i satisfaisant les propriétés suivantes :

précision Si on_border_i est $vrai$, alors i est un noeud de bordure faible.

Complétude Chaque noeud de bordure fort a au moins un voisin j tel que $on\_border_j = vrai$.

Connexité L'intuition se cachant derrière cette propriété est simple : il faut que l'ensemble des noeud de bordure forme une ceinture connexe du réseau. La traduction en terme de propriétés l'est moins : la propriété de complétude ci-dessus ne permet pas d'assurer la connexité. De même, traduire cette propriété en termes de connexité du sous-graphe formé par les noeud détectés en bordure ne marche pas : supposer le graphe 1-connexe, ou 2-connexe n'évite pas le cas du « U » : les noeuds détectés sont tous connectés par le sud, mais il manque un pont par le nord. Il faut pouvoir exprimer le retrait non pas en termes de connexité, mais en termes de zones géographiques.

Une solution est la suivante. Elle a l'inconvénient de réduire fortement le nombre de bordures possibles. Soit $G_b = (V_b, E_b \subset (V_b \times V_b))$ avec $V_b = \{i \in V \text{ t.q. } on\_border_i = vrai\}$. Alors pour tout sous-graphe *connexe* G' de G_b, $G_b - G'$ reste connexe.

Détection de bordure dans des ensembles convexes

A ce jour, nous ne connaissons pas de solution calculant une telle bordure de manière sûre en exploitant simplement les informations de connexité. Plusieurs pistes existent, la principale

difficulté résidant dans la variété des formes que le réseau peut prendre : un réseau en forme de U, c'est-à-dire fortement concave, possède une bordure très difficile à détecter.

Les réseaux convexes, quant à eux, possèdent une propriété plutôt intéressante : ils contiennent toujours leur barycentre. Ceci va permettre d'utiliser des critères de distance pour détecter les noeuds de bordure. L'idée est d'abord présentée dans [RRP+03], où les auteurs affirment détecter des noeuds de bordure à l'aide d'un noeud initiateur. Ce noeud initiateur inonde le réseau afin que tous les noeuds du réseau apprennent leur distance à celui-ci. Dans leur solution, un noeud décide qu'il est en bordure s'il est le noeud le plus loin de l'initiateur parmi ses voisins à deux sauts. Malheureusement, la réussite de cette solution dépend énormément du placement de l'initiateur : s'il est choisi au hasard dans le réseau et que sa position n'est pas au centre de celui-ci, une partie de la bordure ne sera pas détectée. Il est bien sûr dans ce cas possible de faire l'hypothèse d'un initiateur positionné spécialement au milieu du réseau, mais cela complique le déploiement.

Dans [KMR+07], nous présentons une approche plus robuste fondée sur le même principe, bien que plus coûteuse. L'idée est d'utiliser plusieurs initiateurs (moins de 10). Comme précédemment, ces initiateurs inondent le réseau, et chaque noeud apprend sa distance à chaque initiateur. Chaque noeud calcule ensuite sa distance moyenne à l'initiateur, et c'est cette distance qui est exploitée pour la détection de bordure (les noeuds de bordure sont localement les noeuds ayant les distances moyennes les plus grandes). Utiliser plusieurs initiateurs choisis aléatoirement pour calculer une distance moyenne revient à calculer la distance au barycentre des initiateurs. Si l'espace est convexe, alors ce barycentre est toujours situé dans le réseau, ce qui permet le bon fonctionnement de la méthode.

Enfin, citons le cas de [ZZF09] et de [BKT09] qui proposent des méthodes de détection des « trous » du réseau, c'est-à-dire des zones situées à l'intérieur du systèmes ou aucun capteur n'est présent. Malheureusement, dans [ZZF09] la détection suppose la possibilité de mesurer l'angle entre deux capteurs voisins, et dans [BKT09] la détection utilise les positions géographiques réelles des capteurs. Enfin, détecter les trous et détecter la bordure sont deux problèmes proches mais différents, et les solutions présentées ne sont pas applicables en l'état.

5.1.4 Modèle

Nous nous concentrons sur les scénarios où la région surveillée ne permet pas d'intervention humaine. Pour des raisons de simplicité, l'analyse des coûts sera effectuée soit sur une distribution de noeuds uniforme dans une zone de forme rectangulaire, soit en supposant les noeuds répartis régulièrement sur une grille. Le principe demeure le même sur d'autres distributions. Dans le cas d'une grille de n noeuds, chaque rangée et colonne contient $\sqrt{n}$ noeuds.

Noeuds Le système est composé d'un ensemble fini de N capteurs répartis de façon uniforme sur une zone géographique[2]. Chaque noeud possède un identifiant unique i.

Communication Chaque noeud peut communiquer avec tout autre noeud situé à portée radio, qui est modélisée par un disque de diamètre R. On suppose le réseau non partitionné et les communications bidirectionnelles.

Connaissance initiale Initialement, un noeud connaît seulement son identité, le fait qu'il soit le seul à posséder cette identité dans le réseau et un paramètre, d, définissant la dimension de l'espace de coordonnées. De plus, chaque noeud possède un détecteur de bordure possédant les propriétés présentées ci-dessus.

[2]Une fois de plus, pour des raisons de simplicité, la distribution est supposée uniforme. Bien qu'il soit trop tôt pour affirmer que notre algorithme converge indépendamment de la distribution, les simulations que nous avons effectuées sur des distributions plus exotiques vont dans ce sens.

5.1.5 Segmentation de la bordure

`Vincos` repose sur la définition de *segments de bordure* qui vont être utilisés comme référentiel pour le système de coordonnées. Cette partie présente la méthode utilisée pour définir les segments de bordure à partir de la connaissance des noeuds de bordure.

Cette partie se concentre donc sur la bordure. Travailler sur la bordure présente plusieurs avantages que nous avons présentés plus haut. La bordure est à la fois facile et économique à travailler. La facilité de la structuration est due à la forme de « ceinture » (*i.e.* un anneau d'une largeur inférieure à un hop) de la bordure : on va y effectuer une élection de leader sur un anneau. Cette élection va permettre de garantir l'unicité du découpage et permettre de collecter les informations nécessaires au découpage (la taille, en hops, de la bordure). Les messages de cette phase ne sont considérés que par les noeuds de bordure (*i.e.* les noeuds tels que $on\_border = vrai$).

Démarrage de l'élection Chaque noeud i tel que $on\_border_i = vrai$ informe ses voisins qu'il est en bordure. Cela permet à chaque noeud de bordure de découvrir quels sont ses voisins de bordure $V_B(i) \subset V_b$. L'objectif est d'élire le noeud ayant la plus petite identité sur l'anneau de bordure.

Dans cette idée, seuls les noeuds dont les voisins de bordure ont tous des plus grandes identités ont une chance d'être élus leaders de bordure. Afin d'éviter des messages inutiles, seuls ces noeuds rentre en compétition pour l'élection. Les noeuds rentrent en compétition en émettant un message que nous appelons « sonde ». Un noeud j émet une sonde si, et seulement si,

$$\forall i \in V_B(j), j \leq i$$

Progression de la sonde La sonde, une fois émise par un candidat au leadership de la bordure, va suivre un mécanisme de propagation extinction le long de la bordure. Nous cherchons à élire le noeud de plus petite identité. Ainsi, un noeud j va relayer une sonde émise par i si, et seulement si, i est la plus petite identité de candidat qu'il connaît. Sinon, j ignore le message.

A cette fin, lorsque j relaie une sonde, il stocke l'identité du candidat (i). De plus, la sonde contient un compteur de sauts qu'il incrémente avant de réemettre la sonde.

Si plus tard i reçoit une autre sonde créée par j, comprenant une nouvelle distance à j, d'_j, il décide selon la valeur de d'_j. Si $d'_j > d_j + 1$, i ne relaie pas cette nouvelle sonde. Sinon, i peut avoir d'autres tâches à accomplir (voir le paragraphe arrêt de la sonde ci-dessous).

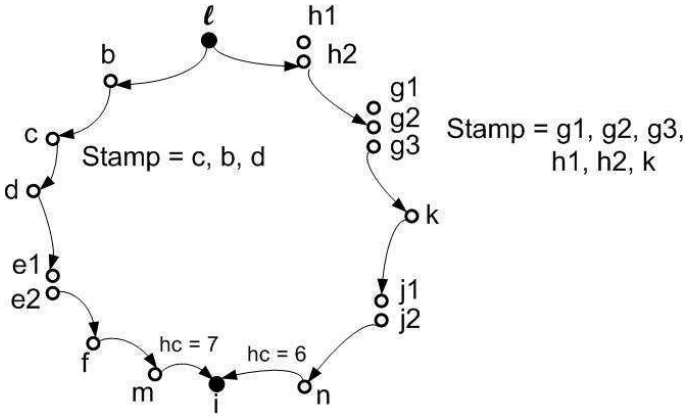


FIG. 5.3 – Chaque sonde transporte le voisinage du deuxième noeud la relayant.

Arrêt de la sonde

Le mécanisme de propagation-extinction va agir comme un filtre. Ainsi, seule la sonde créée par le noeud de V_b ayant la plus petite identité (soit l ce noeud) va parcourir toute la bordure.

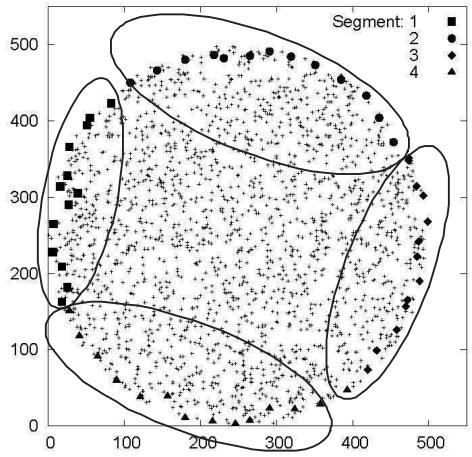


FIG. 5.4 – Les 4 segments définis dans un réseau de 2000 noeuds.

Comme le réseau a une forme en anneau, la sonde progresse selon deux directions (horaire et anti-horaire) jusqu'à l'opposée de l sur l'anneau. Soit $gauche$ et $droite$ ces deux directions.

Un noeud situé à l'opposé de l reçoit ainsi deux sondes de l, provenant de $droite$ et de $gauche$, connaît $1)$ l'identité du leader de bordure, l, et $2)$ la taille (en sauts) de la bordure en sommant le nombre de sauts des deux sondes. Il lui faut néanmoins pouvoir différencier une sonde de direction $gauche$ d'une sonde de direction $droite$.

Pour cela, chaque sonde p émise par le noeud l comporte un champ additionnel, appelé $stamp_p$. Ce champ contient la liste de voisinage du deuxième noeud relayant p (*i.e.* un noeud tel que $d_l = 2$). Chaque noeud relayant la sonde p stocke $stamp_p$. Ainsi, lorsqu'un noeud i reçoit deux sondes p et p', dont les compteurs de sauts sont d_l et d'_l, si $stamp_p \cap stamp_{p'} \neq \emptyset$ alors i sait que les deux sondes ont parcouru la bordure dans la même direction. Il garde alors la plus petite des deux distances d_l et d'_l. Sinon, si $|d_l - d'_l| \leq 1 \wedge stamp_p \cap stamp_{p'} = \emptyset$, alors i sait que l est le leader et que la longueur en sauts de la bordure est de $d_l + d'_l$. Nous appelons i un *leader opposé*. La figure 5.3 illustre la vérification de la direction empruntée par la sonde.

Définition des segments

Lorsque i se découvre leader opposé, il est capable de segmenter la bordure. Soit $hl = d_l + d'_l$ la longueur de la bordure. La taille d'un segment est donc approximativement $hl\%d$. i envoie donc un message m contenant hl ainsi qu'un des deux champs $stamp$ qu'il a reçus. Ce champ $stamp$ va définir le sens de comptage sur la bordure. Ce message est relayé par tous les noeuds de V_b.

Lorsqu'un noeud k reçoit un tel message m, il calcule le segment s_k auquel il appartient. Soit d_l et $stamp$ la distance et le champ collectés par k au passage de la sonde. Si $m.stamp = stamp$, alors $s_k = d_l\%(hl\%d)$, sinon, $s_k = (hl - d_l)\%(hl\%d)$.

L'algorithme décrivant la segmentation de la bordure est décrit figure 5.5. La figure 5.4 présente le résultat d'une séparation de la bordure en 4 segments.

5.1.6 Définition des coordonnées

Une fois que les noeuds de V_b ont appris à quel segment ils appartiennent, ils inondent le système avec un message contenant ce numéro de segment et un compteur de sauts initialisé à 0. Ces messages sont utilisés par chaque noeud du système afin de calculer leur plus petite distance à chaque segment. Cela procure à chaque noeud un d-uplet de coordonnées $(x_1, \ldots, x_d)$. Ces

```
1:  lid ← ∞; dist ← ∞; stamp ← ∅
2:  procedure RECEIVE PROBE(p)
3:    if p.id < lid then /*There is a better leader candidate*/
4:      if p.hc = 2 then /*Set stamp*/
5:        p.stamp ← my_neighbors
6:      lid ← p.id /*Keep probe data*/
7:      dist ← p.hopcount
8:      stamp ← p.stamp
9:      p.hopcount ← p.hopcount + 1 /*Relay*/
10:     broadcast(p)
11:   else if p.id = lid then
12:     if p.hopcount > dist then /*do nothing ... */
13:       return
14:     if |p.hopcount − dist| ≤ 1 ∧ stamp ∩ p.stamp = ∅ then
15:       /*be opposite border leader*/
16:       broadcast(lockMsg(lid, p.hopcount + 1, stamp))
17:     else
18:       dist ← p.hopcount /*Keep probe data*/
19:       stamp ← p.stamp
20: end procedure
21: procedure RECEIVE LOCKMSG(l)
22:   if stamp ∪ l.stamp = ∅ then
23:     segment_number = (dist%(lh%d))
24:   else
25:     segment_number = ((lh − dist)%(lh%d))
26:   broadcast(l)
27: end procedure
```

FIG. 5.5 – Algorithme de la sonde (Probe)

coordonnées peuvent ensuite être exploitées comme des coordonnées classiques, par exemple à des fins de routage ou de structuration.

5.1.7 Analyse de la complexité

Comme décrit dans le modèle, nous supposons pour ces analyses que les noeuds sont répartis selon une grille carrée. Nous considérons ainsi n noeuds répartis en rangs et colonnes de $\sqrt{(n)}$ noeuds. Il y a donc $4\sqrt{n}$ noeuds de bordure forts. On suppose que les noeuds détectés en bordure sont les noeuds de bordure forts.

Le coût en messages de l'élection de leader est dur à évaluer. Il est au moins de $4\sqrt{n}$ dans le cas idéal ou seul le leader effectif envoie une sonde. Ce coût dépend de plus du nombre de candidats leader. Dans notre grille il existe au plus $2\sqrt{n}$ candidats (un noeud sur deux). Dans la pratique, le coût issu des candidatures infructueuses demeure faible.

Après la définition des segments, chaque noeud ne relaie qu'un message par segment (celui provenant du noeud le plus proche qui appartient au segment), soit un coût en $O(dn)$.

La complexité en temps de la segmentation de la bordure est de $2\sqrt{n}$ étapes. Ensuite, la complexité en temps de l'attribution des coordonnées dépend de la plus grande distance dans la grille, soit $\sqrt{2n}$, soit une complexité en temps de $(2 + \sqrt{2})\sqrt{n}$.

5.2 Des coordonnées virtuelles à la structuration géographique

Cette section traite de structuration géométrique, un outil puissant de structuration des réseaux sans fil. Dans un premier temps, nous motivons le problème de la structuration géométrique, puis nous définissons la structuration géométrique, et enfin nous décrivons comment mettre en place cette structuration à partir des coordonnées de `Vincos`.

Dans la littérature, le principal intérêt des coordonnées est le routage. Nous pensons qu'elle présentent de nombreux autres avantages. Ces coordonnées permettent de manipuler le réseau de nombreuses manières différentes. L'idée de la structuration géométrique est de permettre, à partir d'une formule, d'obtenir une structuration logique du réseau permettant d'organiser celui-ci, à des fins par exemple d'agrégation, de clustering ou de gestion d'énergie.

A titre d'exemple, considérons un réseau de capteurs disséminés sur une vaste zone géographique. Il est montré dans [HC07] qu'agréger les données sur une ligne centrale du réseau en vue de leur collecte par un puits mobile constitue une solution efficace et économique de rassembler l'information. Pour cela, il suffit de partitionner les capteurs en trois groupes : le *Nord*, le *Sud* et l'*Equateur*. L'Equateur est une simple ligne droite servant de point de collecte des informations récupérées par tous les capteurs. Périodiquement, un puits mobile (sur un avion par exemple) survole l'équateur et collecte les données. Pour cela, il suffit d'utiliser un système de coordonnées virtuelles à deux dimensions : soient (x_1, x_2) les coordonnées virtuelles d'un noeud i. Le noeud i appartient au groupe Nord si $x_1 < x_2$, au groupe Sud si $x_1 > x_2$ et à l'Equateur si $x_1 = x_2$. La figure 5.6(a) donne un exemple de ce partitionnement pour un réseau de 2000 noeuds.

Une autre idée d'application serait de définir des structures hiérarchiques pour mettre en place des stratégies de type diviser pour régner. Il faut bien sûr que le problème se prête à la mise en place de ces stratégies, mais imaginons le cas d'un problème de coût quadratique ($O(n^2)$). Considérons un réseau divisé en p parties (tel le réseau 5.6(b), ou $p = 3$). Il est possible d'exécuter une instance du problème de base (par exemple, l'élection d'un leader) dans chaque sous-partie, d'un coût $O\big(p(n/p)^2\big)$. Il reste bien sûr ensuite à fusionner les résultats du problème. Dans ce sens, dans [DWB+07], les auteurs démontrent l'avantage de l'utilisation de clusters dans les grands réseaux de capteurs.

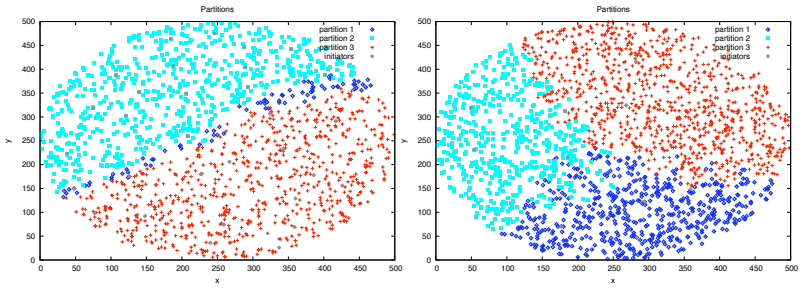
Structuration géométrique

La structuration géométrique peut être définie comme un partitionnement logique du réseau. L'objectif de cette structuration géométrique est de fournir une méthode décentralisée pour procurer à chaque noeud un numéro de partie. Ces parties sont ensuite exploitables par l'application ou le système réparti. Soit $\mathcal{K}$ l'espace de coordonnées des noeuds (dans le cas de `Vincos`, $\mathcal{K} \in \mathbb{N}^d$), et soit p le nombre de parties. Soit respectivement c_i et p_i les coordonnées et la partie du noeud i. Une fonction de structuration géométrique est une fonction f telle que :

$$\begin{array}{rcl} f : \mathcal{K} & \rightarrow & \{0, \ldots, p\} \\ f(c_i) & \rightarrow & p_i. \end{array}$$

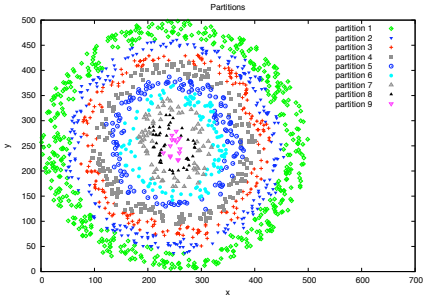
Une telle définition permet à tout noeud de calculer le numéro de partie d'un autre noeud dont les coordonnées sont connues : *chaque noeud peut se représenter la disposition des parties dans le système*. Ce partitionnement peut prendre une forme géographique ou fonctionnelle, comme décrit ci-dessous.

Partitionnement géographique Il s'agit ici de diviser l'espace en zones géographiques adaptées aux besoins de l'application. Il s'agit de diviser le système en parties connexes. Si l'on reprend l'exemple du partitionnement en zones Nord, Sud et Equateur, voici la fonction utilisée (voir figure 5.6(a)) :



(a)

(b)



(c)

FIG. 5.6 – (a)-(c) Exemples de structuration géométrique.

$$\begin{array}{rcl} f : \mathbb{N} * \mathbb{N} & \rightarrow & \{1,2,3\} \\ f(x_1,x_2) & \rightarrow & \begin{cases} 1 & lorsque \quad x_1 > x_2 \\ 2 & lorsque \quad x_1 = x_2 \\ 3 & lorsque \quad x_1 < x_2 \end{cases} \end{array}$$

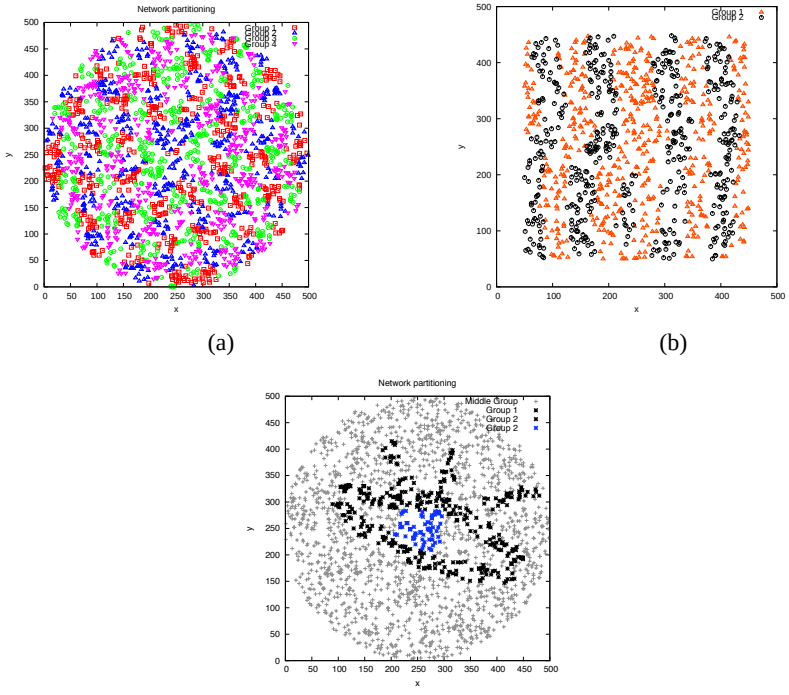
Le partitionnement en cible, comme illustré figure 5.6(c) peut s'avérer utile. Sa définition est très simple à partir des coordonnées `Vincos` en utilisant une seule dimension $d = 1$ (*i.e.* il suffit juste de connaître la bordure). Ce partitionnement est utile à une application de tracking (suivi d'objets mobiles) : lorsqu'aucune cible (ou objet mobile) ne se situe dans le réseau, il est possible de laisser en veille tous les noeuds des anneaux intérieurs ; seule la partie extérieure reste en alerte. Si une cible traverse le réseau, chaque anneau réveille le suivant, le réseau est progressivement réveillé.

$$\begin{array}{rcl} f : \mathbb{N} & \rightarrow & \mathbb{N} \\ f(x_1) & \rightarrow & x_1 \end{array}$$

Partitionnement fonctionnel Dans le partitionnement géographique, à chaque partie correspond une zone spécifique du réseau. Dans le partitionnement fonctionnel, on associe une fonction à chaque partie. Sur le même principe que le partitionnement de l'Equateur, il est possible par exemple de créer des lignes parallèles espacées de j_v sauts comme illustré par la figure5.7(b), en utilisant la formule suivante :

$$\begin{array}{rcl} f : \mathbb{N}^4 & \rightarrow & \{0,1\} \\ f(x_1,x_2,x_3,x_4) & \rightarrow & max(x_1,x_3) \mod j_v \end{array}$$

Similairement, il est possible de créer des lignes verticales espacées de j_h sauts ainsi :



(a) (b)

(c)

FIG. 5.7 – (a)-(c) Partitionnements Fonctionnels : (a) Partitionnement en treillis. (b) Bandes verticales (c) Œil.

$$\begin{array}{rcl} f : \mathbb{N}^4 & \rightarrow & \{0,1\} \\ f(x_1,x_2,x_3,x_4) & \rightarrow & max(x_2,x_4) \mod j_h. \end{array}$$

Enfin, il est possible de combiner ces règles en créant un treillis ($j_h = j_v = 2$) de la manière suivante (voir figure 5.7(a)) :

$$\begin{array}{rcl} f : \mathbb{N}^4 & \rightarrow & \{0,1,2,3\} \\ f(x_1,x_2,x_3,x_4) & \rightarrow & max(x_1,x_3) \mod 2 \\ & & +2 * (max(x_2,x_4) \mod 2) \end{array}$$

Le partitionnement fonctionnel que nous venons de présenter peut être utilisé pour attribuer différents rôles aux noeuds, par exemple afin de répartir les noeuds éveillés et endormis pour économiser l'énergie du système.

Des formes plus compliquées peuvent être utilisées en composant les règles, particulièrement si l'on connaît le diamètre du réseau. La figure 5.7(c) décrit un partitionnement en forme d'œil[3] :

$$\begin{array}{rcl} f : \mathbb{N}^4 & \rightarrow & \{0,1,2,3\} \\ f(x_1,x_2,x_3,x_4) & \rightarrow & \begin{cases} 0 & si & paupières \\ 1 & si & pupille \\ 2 & si & iris \\ 3 & si & cils \\ 4 & sinon, & \end{cases} \end{array}$$

où les prédicats *paupières*, *pupille*, *iris* et *cils* sont définis table 5.1.

[3] Il s'agit juste d'une illustration des capacités de ce système. Nous n'avons pas d'applications spécifiques en tête.

TAB. 5.1 – Prédicats de l'œil

Condition	Description
paupières	$(x_0 = 9 \wedge x_2 < x_0) \vee (x_2 = 9 \wedge x_0 < x_2)$
pupille	$x_1 = x_2 = x_3 = x_4$
iris	$(abs(x_0 - x_2) < 2) \vee (abs(x_1 - x_3) < 2)$
cils	$(x_0 < 12) \wedge (x_1 = x_2 \vee x_2 = x_3 \vee x_1 = x_3)$

Analyse de la complexité Aucun message n'est échangé pour cette attribution de partie. Une fois qu'un noeud a obtenu ses coordonnées, il est en mesure de calculer la partie à laquelle il appartient. Le temps de convergence et le coût en message sont donc nuls.

5.3 Evaluation des performances

Cette section décrit les expériences que nous avons effectuées pour attester de l'efficacité et de la précision de nos propositions. Les expériences ont été conduites avec un simulateur discret écrit en Java. Nous sommes intéressés par l'évaluation algorithmique du protocole, aussi le simulateur n'implémente pas une couche Mac et un modèle de communication réaliste, mais une version simplifiée sans perte de messages, ni collisions.

5.3.1 Coûts de communication

Si l'on ajoute le coût du système de détection de bordure présenté dans [KMR$^+$07], le coût total de communication de `Vincos` est de $O((2 + y + d) * n + 4 * z\sqrt{n})$, n étant nombre total de noeuds, y et z les nombres d'initiateurs et d'initiateurs de bordure (ces termes sont définis dans [KMR$^+$07]), resp., et d le nombre de dimensions du système de coordonnées.

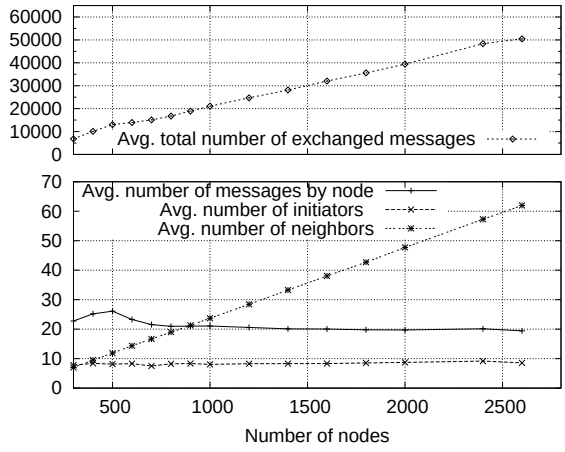


FIG. 5.8 – Analyse du coût en fonction de la taille du réseau, pour $d = 4$, portée radio de 40-unités.

Nous simulons des réseaux comportant 250 à 2600 noeuds. Ces noeuds sont répartis sur une surface plane de 500×500 unités de distance. La zone de portée radio est un disque dont le diamètre varie de 30 à 50 unités de distance.

La figure 5.8 confirme le résultat théorique : le coût d'exécution de `Vincos` par noeud est bas et indépendant de la taille du réseau. On peut observer que même si le nombre total de messages augmente avec la taille du réseau, le coût par noeud décroît légèrement à mesure que le système croît. Ceci est dû au fait que le nombre de messages échangés par les noeuds internes reste constant et faible, mais que leur proportion dans le système augmente. Ainsi plus le système comporte de noeuds, plus la proportion de noeuds de bordure baisse au profit des noeuds internes. Or ce sont les noeuds de bordure qui génèrent la majorité des messages échangés. Donc le coût moyen par noeud baisse[4].

De façon à observer l'impact de la densité sur le coût de `Vincos`, nous avons simulé un réseau de 500 noeuds, dont la portée varie afin de faire varier la densité. La figure 5.9 présente les résultats de cette simulation, en ne considérant que le coût du protocole (c'est-à-dire sans la détection de bordure), et pour $d = 8$ qui est le nombre de dimensions utilisées dans les expériences de routage qui suivent. On peut observer un coût bas, proche de la limite incompressible de d messages par noeud (un message par coordonnée). Le coût par noeud diminue avec la densité pour des raisons semblables à celles énoncées précédemment : l'augmentation de la portée rend la bordure plus petite en nombre de hops et (il s'agit d'un artefact du système de détection de bordure utilisé en amont de `Vincos`) composée de moins de noeuds.

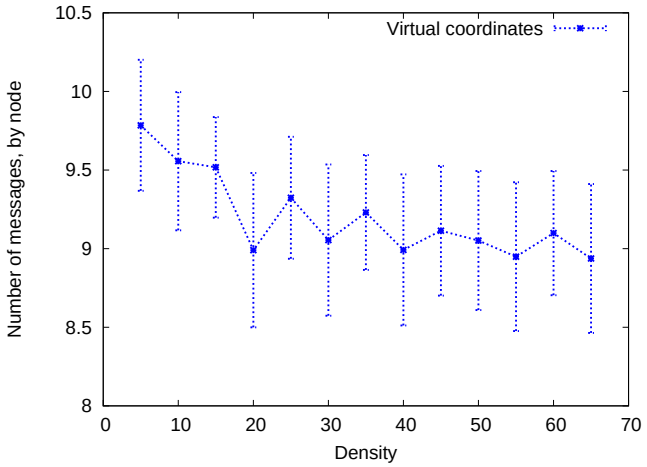


FIG. 5.9 – Coût de Vincos, pour $d = 8$, en fonction de la densité du système.

5.3.2 Précision

Bien que le routage ne soit pas l'objet de cet article, il est intéressant d'observer la précision de `Vincos`. Il est reconnu [BPA+06] qu'un système de coordonnées précis permet un routage glouton efficace. Afin d'observer cette précision et par conséquent les capacités de routage de `Vincos`, nous avons étudié la répartition des noeuds partageant les mêmes coordonnées.

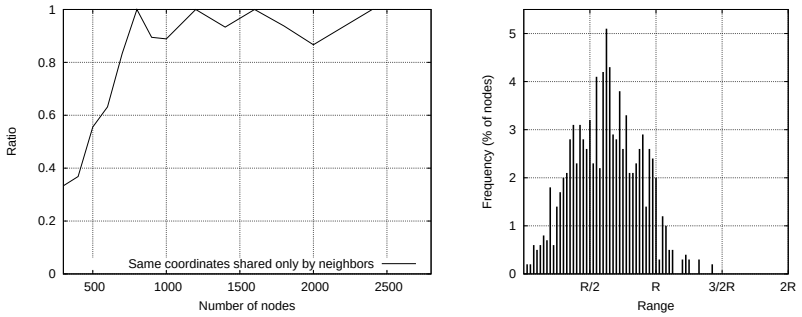
Nous appellerons *système précis* un système vérifiant l'assertion : "si deux noeuds partagent les mêmes coordonnées alors ils sont voisins". Notons qu'un système précis permet un bon routage glouton, puisqu'un message arrivant aux coordonnées finales est arrivé à destination.

Nous avons effectué 20 expériences indépendantes pour différentes tailles de système et une portée radio de 50 unités de distance. La figure 5.10(a) montre, en fonction de la taille du système simulé, quel est le ratio des expériences ayant mené à un système précis. On peut observer que,

[4] Ce n'est pas le cas dans [RRP+03], puisque chaque noeud de bordure inonde le réseau.

pour certaines tailles, tous les systèmes simulés étaient précis. Pour une taille de 1000 noeuds, 90% des systèmes sont précis.

La figure 5.10(b) montre le détail d'un système de 1000 noeuds qui n'est pas précis. Afin de montrer qu'une grosse partie du système est tout de même précise, cette figure représente la répartition des distances physiques entre des noeuds partageant les mêmes coordonnées (des *homonymes*). R est la portée radio (ici, $R = 50$), donc les homonymes distants de plus de R unités ne sont pas voisins. Ainsi, on peut y lire que $3,2\%$ des homonymes sont distants de $R/2$ unités. On observe que la majeure partie des homonymes sont voisins, et qu'il n'y a pas d'homonymes distants de plus de $3/2 \times R$. On peut conclure que, même si tous les homonymes ne sont pas voisins, la majeure partie le sont et qu'un routage glouton dans un tel système ne poserait pas de problèmes.



(a) (b)

FIG. 5.10 – (a) Ratio des simulations ayant conduit à des systèmes précis. (b) Histogramme des distances entre noeuds homonymes dans un système non précis de 1000 noeuds.

5.3.3 Routage et densité

Dans cette partie, nous étudions l'impact de la densité sur la qualité du système de coordonnées de `Vincos`. Un bon système de coordonnées doit être utilisable pour acheminer les messages de leur source à leur destination. Pour cela l'algorithme le plus simple, appelé *routage glouton*, consiste à relayer le message à transmettre au voisin dont les coordonnées se rapprochent le plus de la destination. De bonnes coordonnées assurent que la majorité des messages ainsi échangés arrivent à destination.

Nous comparons les taux d'échec de transmission de messages à l'aide d'un algorithme glouton sur les coordonnées virtuelles (avec $d = 8$) et parallèlement, pour référence, du même algorithme sur les coordonnées physiques réelles des noeuds. Pour chaque test, nous simulons ainsi l'échange de 10^4 messages entre deux noeuds du réseau choisis aléatoirement.

Ici, 500 noeuds sont répartis uniformément aléatoirement dans un espace de 500×500 unités de distance. Pour faire varier la densité, nous faisons varier la porté des noeuds : de 30 à 100 unités de distance. La figure 5.11 représente les résultats de cette expérience. La courbe pointillée représente le taux d'échec de routage en utilisant les coordonnées physiques, et la courbe pleine représente le même taux en utilisant les coordonnées `Vincos`. Chaque point est la moyenne d'environ 25 expériences. Les deux courbes sont légèrement décalées afin de faciliter la lecture des écarts types.

Cette figure présente d'intéressants résultats : le routage sur coordonnées virtuelles se comporte mieux que le routage sur coordonnées physiques en petites densités. C'est un bénéfice connu des coordonnées virtuelles : les faibles densités sont souvent la source de micros "trous" dans la topologie du réseau. Les coordonnées physiques sont établies indépendamment de ces trous, et les

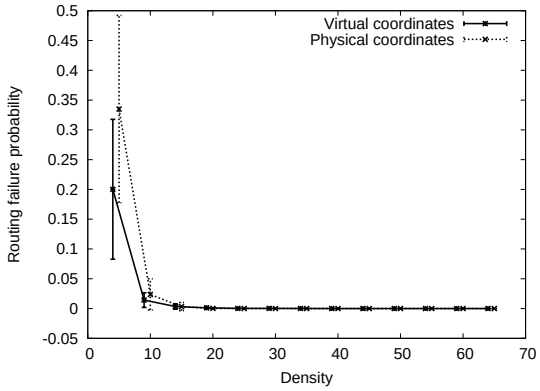


FIG. 5.11 – Impact de la densité sur le taux de succès du routage.

messages se retrouvent souvent arrêtés par la présence de ces trous s'il utilisent les coordonnées physiques. A l'opposée, les messages permettant à chaque noeud de découvrir ses coordonnées virtuelles ont eux été affectés par les trous. Les coordonnées virtuelles reflètent alors ces pièges et les messages transitent mieux en utilisant celles-ci.

Il est difficile à distinguer sur la figure que les coordonnées physiques donnent de meilleurs résultats que les coordonnées virtuelles avec une densité de 20 noeuds par voisinage (Physiques : $3\,{}^{0}/_{00}$ messages perdus, Virtuelles : $12\,{}^{0}/_{00}$). Cela provient probablement des quelques cas ou `Vincos` n'est pas totalement précis alors que la densité suffisante permet au routage sur coordonnées physiques de fonctionner pleinement. Dès que la densité augmente (30 noeuds par voisinage), `Vincos` s'améliore ($2\,{}^{0}/_{00}$ contre $0\,{}^{0}/_{00}$) et à une densité de 40 noeuds par voisinage, les routages des deux systèmes permettent une livraison à coup sûr.

5.3.4 Routage et pertes

Nous allons ici étudier l'impact des pertes de messages et de la forme du réseau sur `Vincos`. Encore une fois, ces impacts seront évalués indirectement par la fiabilité du routage mis en oeuvre sur les coordonnées virtuelles.

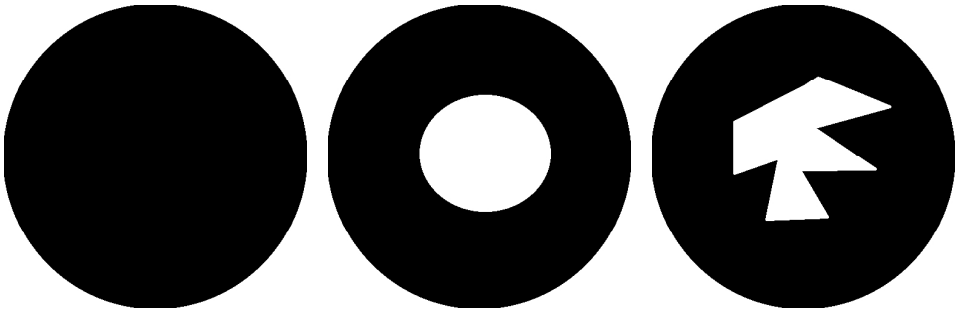
Pour étudier l'impact de la forme du réseau, nous ne répartissons plus les noeuds sur la totalité de l'aire de simulation : nous les répartissons uniformément selon les trois masques présentés ci-dessous. L'algorithme de placement des capteurs fonctionne ainsi : une position aléatoire est uniformément tirée, si cette position est dans la zone noire du masque, disposer le capteur à cet endroit, sinon recommencer.

Afin d'étudier l'impact des pertes, nous avons modélisé les pertes de messages par la fonction f de pertes suivante (avec $Pmin = 0.3$, et d la distance physique entre l'émetteur et le récepteur du message) :

$$f(d) = \begin{cases} 1 & \text{si } d < r \\ 0 & \text{si } d > R \\ Pmin - \sqrt{\frac{R-d}{R-r}} * (\frac{R-d}{R-r} - 5) * \frac{1-Pmin}{4} & \text{sinon.} \end{cases}$$

Informellement, cela revient à attribuer deux portées aux noeuds. Une portée sûre, r, en dessous de laquelle toute communication est reçue, et une portée maximale, R, au delà de laquelle aucune communication n'est reçue. Entre les deux portées, la réception est aléatoire.

Afin d'obtenir une densité moyenne de 24.5 noeuds par voisinage (moyenne sur les trois masques), nous avons posé $R = 60$ et $r \in 40, 50, 60$. Étant donné que nous n'utilisons ni de



(a) (b) (c)

FIG. 5.12 – (a) Réseau circulaire plein (b) Réseau circulaire - trou circulaire (c) Réseau circulaire - trou irrégulier.

couche mac, ni de système d'acquittement et de réémission, les messages perdus ne sont pas réémis. La figure 5.13 illustre les taux de pertes théoriques en fonction de la distance, obtenus avec les paramètres décrits ci-dessus.

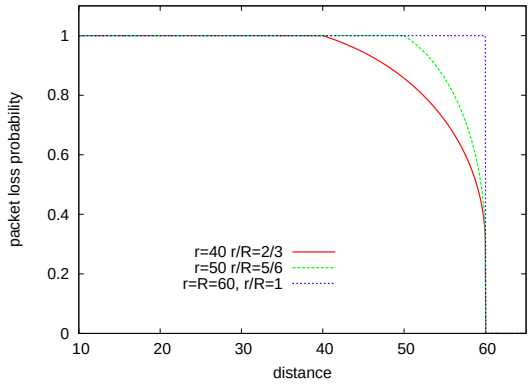


FIG. 5.13 – Taux de perte moyens en fonction de la distance émetteur-récepteur et de r.

Notons que la transmission des paquets de test pour le routage s'effectue sans perte (puisqu'il serait alors difficile de différentier l'impact des pertes sur la construction de `Vincos` de l'impact des pertes sur le routage lui-même). Néanmoins les découvertes de voisinages et tous les protocoles précédant la transmission des paquets de test pour le routage s'effectuent avec des pertes. Comme précédemment, l'algorithme de routage est le routage glouton sur les coordonnées virtuelles et pour contrôle sur les coordonnées physiques réelles.

La figure 5.14 présente les résultats de ces simulations. Chaque paire de barres présente à gauche les taux de pertes pour les coordonnées virtuelles, à droite les taux de pertes pour les coordonnées physiques. Chaque groupe comporte 4 paires, indiquant de gauche à droite les taux d'échec de routage pour :

- les masques circulaire plein 5.12(a) et circulaire avec trou circulaire 5.12(b) réunis (nommés « Virtual-all » et « Physical-all »).
- le masque circulaire plein 5.12(a).
- le masque circulaire avec trou circulaire 5.12(b).
- le masque circulaire avec trou irrégulier 5.12(c).

Cette figure amène plusieurs constats. Le plus frappant est le cas du masque irrégulier, où le

routage sur coordonnées virtuelles surpasse largement le routage sur coordonnées physiques, qui culmine à au moins 15% d'échecs de routage. Ceci est dû à la forme du trou du masque 5.12(c), qui offre de multiples pièges pour les messages souhaitant traverser le système. Comme expliqué précédemment, les messages d'attribution des coordonnées sont eux aussi affectés par la forme du système, et ainsi les coordonnées virtuelles se comportent mieux.

Deuxième constat, les deux routages, hors le cas irrégulier, offrent des performances comparables. Les coordonnées virtuelles sont légèrement moins efficaces que les coordonnées physiques dans le cas d'un réseau sans trou, mais elles sont plus efficaces dans le cas d'un trou circulaire. Ceci confirme la capacité de `Vincos` à proposer un routage efficace et à gérer les topologies trouées.

Enfin, lorsque r augmente, le taux d'échec de routage diminue pour les deux systèmes de coordonnées. Ceci montre que `Vincos`, bien qu'un peu affecté par les pertes de messages, résiste plutôt bien à ceux-ci. Le routage sur coordonnées physique s'améliore car chaque noeud découvre plus de voisins lorsqu'il y a moins de pertes de messages.

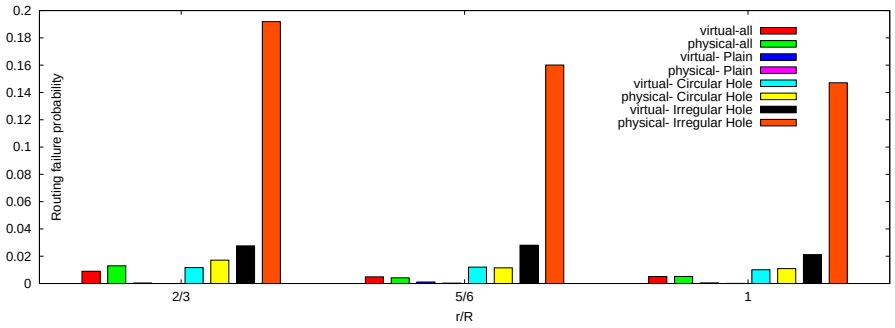


FIG. 5.14 – Taux d'échec du routage en fonction des pertes et de la forme du réseau.

5.3.5 Taille des parties

Dans cette expérience, nous observons l'influence de la densité du réseau sur la distribution des parties. Afin d'estimer quantitativement la qualité du partitionnement réalisé, nous étudions la taille relative des parties : après l'exécution de `Vincos`, nous partitionnons le système en parties de tailles égales. Soit n_{min} et n_{max} le nombre de noeuds composant respectivement la plus petite et la plus grande partie (en nombre de membres). Nous calculons alors le rapport $\frac{n_{min}}{n_{max}}$. Si ce rapport est proche de 1, le partitionnement est de bonne qualité et à l'extrême un rapport de 1 signifie que toutes les parties ont la même taille.

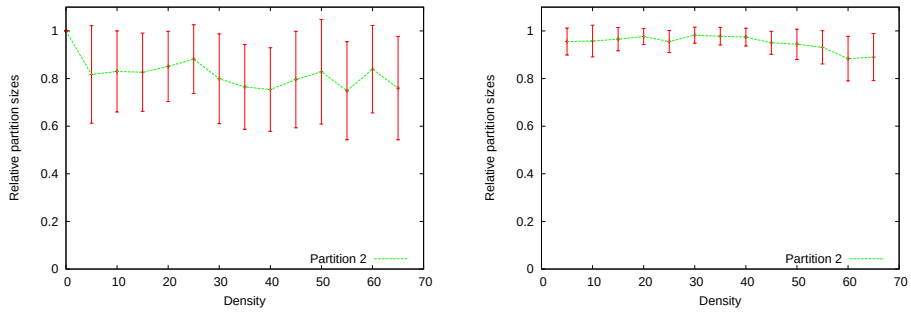
Nous avons utilisé deux prédicats de partitionnement :

– un partitionnement géométrique "en camembert" produisant un partitionnement semblable au partitionnement figure 5.6(b) et défini par la fonction

$$\begin{array}{rcl} f : \mathbb{N}^4 & \rightarrow & \{0,1,2,3\} \\ f(x_{1..4}) & \rightarrow & i \textit{ tel que } x_i = min(x_{1..4}) \end{array}$$

– un partitionnement fonctionnel produisant un partitionnement semblable au partitionnement figure 5.7(b), avec $j_h = 2$.

Ces deux figures illustrent la qualité du partitionnement effectué. Sur la figure 5.15(a), on peut observer que la plus petite des parties fait en moyenne 80% de la grande partie, quelque soit la densité. Si l'on considère la métrique employée, il s'agit d'un bon ratio : puisque le réseau comporte 500 noeuds, la plus mauvaise répartition amenant à un tel rapport de 80% serait la suivante : $\{104, 132, 132, 132\}$.



(a) (b)

FIG. 5.15 – (a) Qualité du partitionnement géographique (b) Qualité du partitionnement fonctionnel.

La figure 5.15(b), élaborée à partir d'un partitionnement fonctionnel, donne d'encore meilleurs résultats, puisque le rapport se situe ici autour de 95% pour les densités inférieures à 50 noeuds par voisinage, puis décroît vers les 90% pour 70 noeuds par voisinage. La dégradations des performances avec de hautes densités s'explique : pour faire varier la densité, nous augmentons la portée radio. Plus la portée radio est forte, plus la longueur d'un saut est grande. Or le saut est notre mesure des distances : plus le saut est grand moins la mesure est précise, et moins le partitionnement est précis.

5.4 Conclusion

Ce chapitre contient une étude, publiée dans sa version originale [KMR+09]. Dans ce chapitre, nous nous concentrons sur une application particulière : la création d'un système de coordonnées virtuelles. C'est un chapitre important dans notre tour d'horizon des structures de communication puisque c'est la structure de communication elle-même qui est capturée et synthétisée, à l'aide d'un système de coordonnées. Il s'agit de la transformation d'une structure physique (la structure de communication sous-jacente) en une structure logique (les relations entre les coordonnées de noeuds voisins). Cette structure logique a l'avantage d'être explicite, ce qui permet à des noeuds de connaître leurs positions respectives et par exemple de router des messages. Ce chapitre apporte les contributions suivantes :

- Nous proposons une définition formelle de la « bordure » d'un système. Nous pensons que l'intérêt de la définition et la détection d'une telle bordure dépasse largement le cadre des systèmes de coordonnées virtuelles. La bordure représente en effet un sous-ensemble du réseau ayant des propriétés très utiles : elle entoure le réseau et elle est étroite ce qui y facilite les consensus. Enfin, épuiser les noeuds de la bordure est rarement problématique. Nous définissons ensuite le détecteur de bordure imparfait.
- Utilisant un tel détecteur de bordure imparfait, nous présentons `Vincos`, un algorithme permettant d'attribuer à chaque noeud des coordonnées dans un espace virtuel. Bien que ce protocole ne soit pas le premier reposant sur l'utilisation de la bordure, il l'exploite à très faible coût. L'idée est de commencer par ne structurer que la bordure et d'ensuite diffuser le résultat.
- Par des simulations, nous testons et comparons la qualité du système de coordonnées produit par rapport au système de coordonnées physiques. Nous montrons que dans certains cas, utiliser les coordonnées virtuelles est plus efficace qu'utiliser les coordonnées physique pour acheminer un paquet dans le réseau. Nous montrons que même lorsqu'il y a des pertes,

`Vincos` arrive à attribuer des coordonnées virtuelles utiles aux noeuds.

– Comme application des systèmes de coordonnées virtuelles, nous introduisons l'idée de structuration géométrique du réseau. Nous définissons l'idée assez naturelle de partitionnement géographique et fonctionnel du réseau et montrons comment l'appliquer aux coordonnées obtenues par `Vincos`. Des simulations montrent que cette approche est efficace et permet la définition de parties équilibrées.

Chapitre 6

Conclusion

6.1 Contexte

Cette étude a porté sur les systèmes répartis. Nous avons vu en introduction les raisons menant à l'émergence des systèmes répartis :

- La possibilité. La multiplication des entités autonomes capables de calcul, leur miniaturisation et leur omniprésence (grossissent les rangs du monde réparti en construction) créent énormément d'entités disponibles pour un système réparti. Conjointement, on assiste à une progression des mécanismes d'interconnexion de ces machines : les débits et l'accessibilité des connexions filaires ont considérablement augmenté pour les particuliers comme pour les entreprises. Récemment, la mise en réseau d'entités c'est libérée des fils avec l'arrivé du `Wifi` et des communications `GSM`.
  Ces raisons, que l'on peut mettre sous l'étiquette du progrès de la technique, ont augmenté le nombre de systèmes dans lequels il est possible de créer des systèmes répartis.
- Le besoin, c'est-à-dire le changement des attentes des utilisateurs vis-à-vis du monde numérique. Plus exactement, il faudrait parler de la multiplication de ces attentes. Le monde numérique se voit confier de plus en plus de tâches : des données importantes (enregistrements bancaires, photos de famille, dossiers médicaux), des décisions importantes (aiguiller des trains, dresser des procès verbaux, trouver un itinéraire). Ces tâches nécessitent la fiabilité des systèmes impliqués. Comme nous l'avons vu dans l'introduction, les systèmes répartis proposent dans ce cas une solution attrayante : ils sont plus robustes puisqu'ils ne sont pas conçus autour d'une unique entité centrale faillible.
  La multiplication des entités numériques, de leurs interactions et du nombre d'utilisateurs de tels réseaux pose aussi des problèmes de passage à l'échelle. Ainsi, quand bien même nous serions capables de construire une unique entité parfaitement fiable pour effectuer une tâche donnée, cette entité devrait de plus être très puissante afin de traiter des requêtes chaque jour de plus en plus nombreuses. Ici aussi, les systèmes répartis séduisent par leur capacité à s'adapter à la quantité de demandes des utilisateurs : comme l'illustre la tendance actuelle de multiplication des coeurs dans les processeurs, mettre plusieurs entités là où une seule ne suffit plus constitue une réaction assez logique.
  Ces raisons ont rendu l'informatique répartie désirable, voire nécessaire.

La conjonction de la possibilité de créer de tels systèmes et de la nécessité d'évoluer vers des systèmes possédant ces propriétés explique le succès de l'approche répartie. Cependant nous avons aussi vu en introduction que le passage en réparti a un coût.

Ce coût est celui d'une complexité accrue : il est difficile de comprendre, de réfléchir et de prévoir les phénomènes ayant lieu dans un système réparti. Ceci rend la conception de tels systèmes peu aisée. De plus, les systèmes répartis désignent un domaine vaste : il y a finalement

peu de rapport entre des processeurs partageant une mémoire et des téléphones portables regroupés à un arrêt de métro. Face à cette difficulté, la méthode et la science ont développé des stratégies classiques.

C'est ici qu'interviennent les modèles. L'objectif d'un modèle est d'offrir une prise à la théorie en capturant un ensemble de systèmes partageant des propriétés communes. C'est ainsi que fut développée une hiérarchie de modèles pour les systèmes répartis. Un système réparti étant constitué d'entités collaborantes, il existe des modèles pour les entités et des modèles pour leur collaboration.

Dans cette étude nous nous sommes intéressés aux modèles de collaboration des entités. Une caractéristique intrinsèque des systèmes répartis étudiés est l'absence de connaissance globale : toutes les entités ne communiquent pas entre elles (pour deux raisons : l'asynchronie et le voisinage limité). Au contraire, chaque entité partage de l'information avec un petit nombre d'autres entités. C'est ce qui crée une structure des échanges d'informations reliant les entités selon qu'elles partagent de l'information ou non. C'est sur ces structures que porte cette étude.

6.2 Résumé des chapitres

Partant de cette notion de modèle et du constat qu'un modèle induit une structure (au sens de lien entre les entités), notre étude a suivi un parcours que nous allons reprendre dans cette partie.

La notion de modèle abordée dans l'introduction (chapitre 1) est détaillée dans le chapitre 2. Cet approfondissement se fait par la présentation de différents modèles classiques de systèmes répartis. Cela nous a permis d'un côté, de découvrir la grande diversité de ces systèmes et de leurs applications, mais de l'autre de comprendre aussi leurs similitudes. Cela fut aussi l'occasion d'arpenter la jonction entre la théorie et la pratique : nous y avons vu qu'un modèle utile théoriquement ne l'est pas forcément dans la pratique et nous avons introduit les notions de complexité qui, au-delà de la solvabilité théorique d'un problème, impactent l'utilité pratique d'un résultat. Enfin, grâce à l'analogie de la maison, nous avons exploré la philosophie de conception des systèmes répartis : entre robustesse, économie et facilité de mise en oeuvre, l'ingénieur choisit des matériaux adaptés à son problème. En amont, la recherche développe ces matériaux, étudie leurs interactions et modélise l'environnement dans lequel ils vont être déployés. Ainsi, l'objectif de cette étude n'est pas de construire, mais de développer les matériaux utiles (souvent appelés briques de base) pour la construction.

Cette étude se concentre ensuite, dans le chapitre 3, sur l'étude de l'impact des structures. Toujours dans l'analogie de la maison, nous étudions l'impact des fondations sur la robustesse du bâtiment. "En général, un modèle contient seulement les ingrédients essentiels que nous devons considérer afin de comprendre et résonner sur certains aspects du comportement d'un système" [CDK05]. Dans cette partie, au travers d'un brique de base (l'évaluation du nombre d'entités vivantes dans le système), dans le modèle classique des systèmes à passage de messages, nous obtenons des comportements radicalement différents (et parfois marginalement compréhensibles) selon la structure de communication sous-jacente. Cette structure de communication a donc un impact considérable sur la performance et la fiabilité des applications l'utilisant comme fondation. Nous montrons ensuite qu'il est possible, puisque l'impact est mesurable, de déployer un protocole évaluant l'état de cette couche de communication, dans le contexte des grands graphes. Ce chapitre nous a permis de mettre en lumière et d'étudier l'impact de la couche de communication sous-jacente.

Puisque que la structure de communication sous-jacente a un impact, nous étudions dans le chapitre 4 comment exploiter, réduire, manipuler cet impact. Autrement dit, nous rentrons dans le modèle du système des hypothèses sur la structure des échanges d'informations (que nous appelons souvent structure physique) et montrons qu'il est alors possible de réaliser certaines tâches

impossibles sinon. Cette démonstration s'effectue dans le modèle, très abstrait, des systèmes asynchrones à passage de messages soumis aux défaillances byzantines. Cette démonstration s'effectue ensuite dans le cadre réseaux de capteurs, afin de générer des secrets (ou des clés) partagés par des paires de capteurs afin de leur permettre d'échanger confidentiellement de l'information. Alors que dans la première étude nous contraignons le modèle (en ajoutant des hypothèses de synchronie), dans la deuxième nous ne contraignons pas le modèle par des hypothèses structurelles, mais au prix d'un résultat non garanti. Nous montrons cependant qu'il est possible d'énoncer des contraintes structurelles suffisantes pour obtenir des garanties (la différentiabilité locale). Nous transformons dans ce chapitre des contraintes physiques (les messages sont suffisamment rapides, les noeuds possèdent de bonnes configurations de voisinage) en contraintes logiques (les processus décident tous la même valeur, seul deux noeuds possèdent la même clé).

Enfin, tandis qu'au chapitre précédent nous nous arrêtons à l'obtention de la contrainte logique désirée, dans le chapitre 5 nous cherchons à capturer par des contraintes logiques l'ensemble des contraintes physiques du système. Autrement dit, nous cherchons à transformer la structure sous-jacente (ensemble de contraintes physiques sur les flux d'information) en une structure logique explicite (ici, un système de coordonnées). De cette approche résulte un système de coordonnées virtuelles : les coordonnées attribuées aux noeuds dans ce système n'ont aucun rapport avec des grandeurs physiques et découlent uniquement de la capacité des noeuds à communiquer. De façon intéressante, ce système de coordonnées est obtenu à faible coût grâce à une construction en deux temps. La première partie, chère, n'est effectuée que sur un sous-ensemble particulier des noeuds (la bordure du système). La deuxième partie se contente de propager ces résultats partiels à l'ensemble des noeuds. Nous montrons ensuite une exploitation possible de ce système de coordonnées (la structuration géométrique).

6.3 Perspectives et défis

Finalement, cette étude pose plus de questions qu'elle n'en résout. Nous allons tenter d'en énoncer quelques unes dans cette section.

Améliorer les outils

Une première série de questions a trait à l'impossible exhaustivité de ce document. Chaque expérience, chaque modèle, chaque solution mérite plus d'approfondissement. Chaque protocole présenté pose la question de son adaptabilité à un contexte différent, question qu'il est bien souvent frustrant de ne pas avoir le temps d'aborder. Comment adapter un calcul de centralité à un réseau dynamique ? Est-il possible d'adapter le consensus byzantin à un modèle de capteurs mobiles ? Comment utiliser l'isolement de bordure sur des réseaux quelconques ?

Dans ce document, nous avons souvent eu recours à la simulation de systèmes répartis. Cet outil, la simulation, nécessite la définition arbitraire de nombreux paramètres du système. Il est impossible de tester toutes les combinaisons de ces paramètres, ce qui laisse de nombreuses questions en suspens. Améliorer et faciliter la simulation de systèmes répartis est un défi, d'autant que c'est souvent la seule façon de comprendre et de reproduire les phénomènes se déroulant dans les réseaux large échelle. C'est aussi, avec la preuve, une manière fréquente de convaincre, malgré qu'il soit très difficile, voir impossible, de répéter les expériences.

La preuve par l'analyse mathématique a elle l'avantage d'être rigoureuse. Elle est cependant difficilement applicable dans les systèmes répartis de grande taille, ou dans les réseaux mobiles : le caractère dynamique de ces systèmes complexifie grandement les modèles et nous manquons de "poignées" théoriques solides pour y travailler. Les approches récentes, comme la théorie spectrale, ouvrent des portes dans la compréhensions des phénomènes dans les grands réseaux d'interaction qu'il sera intéressant d'explorer.

Comprendre et modéliser les réseaux complexes d'interaction

Dans notre analyse des structures, nous avons tour à tour pris un point de vue tantôt local (centré sur un processus), tantôt global (l'ensemble du système). En physique nous parlerions d'échelle microscopique et macroscopique. Dans l'étude des fluides, ces deux échelles ne suffisant pas, on assiste à l'introduction d'une nouvelle échelle, intermédiaire : l'échelle mésoscopique. Il s'agit d'une échelle intermédiaire, située par exemple entre la molécule d'eau et l'écoulement du fluide. Posséder une telle échelle (*i.e.* la maîtriser théoriquement) pour les systèmes répartis ouvrirait la porte à de nombreuses avancées. Cependant, comme pour le cas physique, cette échelle mésoscopique est plus difficile à définir que les deux autres : l'échelle microscopique et macroscopique sont faciles à définir, puisqu'elles sont respectivement la plus petite et la plus grande résolution (*i.e.* l'unité et le tout). Néanmoins, la détection et l'amélioration du concept de communautés ou encore de motifs (associations répétitives de quelques entités [GL08]) pourrait s'avérer une piste fructueuse en fournissant cette échelle manquante.

De manière générale, la détection de communautés ou de motifs, parallèlement à la caractérisation spectrale des graphes, va dans le sens de la compréhension des structures complexes que nous avons abordées. L'idée que les graphes sont aléatoires a laissé place au modèle d'attachement préférentiel, ce qui témoigne de l'avancée de la compréhension dans le domaine : les phénomènes, une fois compris, ne paraissent plus aléatoires. Ce modèle d'attachement préférentiel se raffine [PDFV05] et c'est le progrès de ces modélisations qui fournira probablement les solutions aux problèmes des grands réseaux d'interaction.

Il est intéressant d'observer qu'intrinsèquement la formation des réseaux d'interaction de machines est étroitement liée aux interactions de leurs possesseurs (proximité physique des utilisateurs de téléphone mobile, goûts communs des mélomanes). Les deux structures (informatique et sociale) sont ainsi étroitement liées. La compréhension de l'une des structures semble une direction intéressante pour comprendre et améliorer les phénomènes de l'autre structure. Ce couplage ouvre de nouvelles applications, comme celle de la recommandation de contenu (de l'informatique vers le social), ou celle de la mise en cache intelligente dans les réseaux pair-à-pair (du social vers l'informatique) dont les promesses sont excitantes.

Index

t-resilients, 8

asynchrones, 7

boite noire, 9
broadcast, 13
byzantin, 50
byzantins, 8

churn, 6
connaissance externe, 78
connaissance interne, 78
correct, 15
correctes, 8
crash-stop, 8
critiques, 29

détecteurs de défaillances, 9
déterministe, 10
distribution stationnaire, 33
dynamique, 6

fifo, 9
fort, 9

indulgents, 7

leader opposé, 85

marche aléatoire, 29
modèle, 3

noeud, 6
noeud de bordure faible, 82
noeud de bordure fort, 82
non déterministes, 10

partiellement synchrones, 51
puits, 60

ronde, 7
routage glouton, 92

statique, 6
structure, 11
synchrones, 7
système réparti, 2

vivant, 15
vivante, 8
voisin, 6
voisins, 13

wait-free, 8

Bibliographie

[AB02] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[ACK+97] Marcos Kawazoe Aguilera, Wei Chen, Marcos Kawazoe, Aguilera Wei, and Sam Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication, 1997.

[ADGFT04] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 328–337, New York, NY, USA, 2004. ACM.

[ADGFT06] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Consensus with byzantine failures and little system synchrony. *Dependable Systems and Networks, International Conference on*, 0:147–155, 2006.

[AF] David Aldous and James Allen Fill. Reversible markov chains and random walks on graphs.

[Ang80] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 82–93, New York, NY, USA, 1980. ACM.

[APS03] Haowen Chan Adrian, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *In IEEE Symposium on Security and Privacy*, pages 197–213, 2003.

[AR07] Antonio Fernández Anta and Michel Raynal. From an intermittent rotating star to a leader. In Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal, editors, *OPODIS*, volume 4878 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2007.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[Bar04] Marc Barthelemy. Betweenness centrality in large complex networks. *EUR.PHYS.JOUR.B*, 38:163, 2004.

[BCBT96] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg. Crash failures vs. crash + link failures. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, page 246, New York, NY, USA, 1996. ACM.

[BDDT07] Joffroy Beauquier, Sylvie Delaët, Shlomi Dolev, and Sébastien Tixeuil. Transient fault detectors. *Distributed Computing*, 20(1):39–51, 2007.

[BDFG03] Romain Boichat, Partha Dutta, Svend Frölund, and Rachid Guerraoui. Deconstructing paxos. *SIGACT News*, 34(1):47–67, 2003.

[BFAF05] F. Benbadis, T. Friedman, M. D. Amorim, and S. Fdida. Gps-free-free positioning system for sensor networks. In *Proc. of WOCN*, April 2005.

[BKT09] Marin Bertier, Anne-Marie Kermarrec, and Guang Tan. Visibility-graph-based shortest-path geographic routing in sensor networks. In *INFOCOM*, 2009.

[BO83] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, New York, NY, USA, 1983. ACM.

[BOCB07] F. Benbadis, K. Obraczka, J. Cortès, and A. Brandwajn. Exploring landmark placement strategies for self-organization in wireless sensor networks. In *Proc. of IEEE PIMRC*, September 2007.

[Bon72] P Bonacich. Factoring and weighting approaches to status scores and clique identification. *J. Math. Sociol. 2*, pages 113–120, 1972.

[BP07] Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, pages 2303–2318, 2007.

[BPA+06] F. Benbadis, J.-J. Puig, M. D. Amorim, C. Chaudet, T. Friedman, and D. Simplot-Ryl. JUMP: enhancing hop-count positioning in sensor network using multiple coordinates. Technical report, TR arXiv cs.NI/0604105, April 2006.

[BPBRT09] Lélia Blin, Maria Gradinariu Potop-Butucaru, Stephane Rovedakis, and Sébastien Tixeuil. A new self-stabilizing minimum spanning tree construction with loop-free property. Technical report, May 2009.

[BR09] François Bonnet and Michel Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash and anonymity. In *DISC*, pages 341–355, 2009.

[Bra01] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[BW04] R. Bischoff and R. Wattenhofer. Analyzing connectivity-based, multi-hop ad hoc positioning. In *Proc. of IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, March 2004.

[CA06] Yi Cheng and D. P. Agrawal. Improved pairwise key establishment for wireless sensor networks. In *WIMOB '06: Proceedings of the 2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 442–448, Washington, DC, USA, 2006. IEEE Computer Society.

[CCDU05] A. Caruso, S. Chessa, S. De, and A. Urpi. Gps free coordinate assignment and routing in wireless sensor netwoks. In *Proc. of IEEE Infocom*, pages 150–160, March 2005.

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, December 1990.

[CDK05] Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design (3rd Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[CF99] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):642–657, 1999.

[CHC04] Denis Caromel, Ludovic Henrio, and Luca Cardelli. *A Theory of Distributed Objects*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[Cin75] E. Cinlar. *Introduction to stochastic Processes*. Prentice Hall, New-Jersey, 1975.

[CKM00] David W. Carman, Peter S. Kruus, and Brian J. Matt. Constraints and approaches for distributed sensor network security. Technical Report 010, NAI Labs, The Security Research Division Network Associates, Inc., September 2000.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[Cri91] Flavin Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, 1991.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[DDS87] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34:77–97, 1987.

[DGFG02] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. A realistic look at failure detectors. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 345–353, Washington, DC, USA, 2002. IEEE Computer Society.

[DGG02] Assia Doudou, Benoît Garbinato, and Rachid Guerraoui. Encapsulating failure detection: From crash to byzantine failures. In *da-Europe '02: Proceedings of the 7th Ada-Europe International Conference on Reliable Software Technologies*, pages 24–50, London, UK, 2002. Springer-Verlag.

[DGGS99] Assia Doudou, Benoît Garbinato, Rachid Guerraoui, and André Schiper. Muteness failure detectors: Specification and implementation. In *EDCC-3: Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, pages 71–87, London, UK, 1999. Springer-Verlag.

[DGV05] P. Dutta, R. Guerraoui, and M. Vukolic. Best-case complexity of asynchronous byzantine consensus, 2005.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DPG01] L. Doherty, K. S. J. Pister, and L. El Ghaoui. Convex position estimation in wireless sensor networks. In *Proc. of IEEE Infocom*, pages 1655–1663, 2001.

[DPMM06] Roberto Di Pietro, Luigi V. Mancini, and Alessandro Mei. Energy efficient node-to-node authentication and communication confidentiality in wireless sensor networks. *Wirel. Netw.*, 12(6):709–721, 2006.

[DWB+07] M. Dohler, T. Watteyne, D. Barthel, F. Valois, and J.-L. Lu. Kumar's, zipf's and other laws: How to structure an optimum large-scale wireless (sensor) network? In *13th European Wireless Conference*, Paris, France, April 2007.

[EG02] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 41–47. ACM Press, 2002.

[ER59] P. Erdos and A. Renyi. On random graphs. In *Publicationes Mathematicae*, pages 6: 290–297, 1959.

[Fei95a] Uriel Feige. A tight lower bound for the cover time of random walks on graphs, random structures and algorithms 6. In *Random Structures and Algorithms*, pages 433–438, 1995.

[Fei95b] Uriel Feige. A tight upper bound on the cover time for random walks on graphs. *RSA: Random Structures & Algorithms*, 6, 1995.

[Fei96] Uriel Feige. A fast randomized logspace algorithm for graph connectivity. *Theor. Comput. Sci.*, 169(2):147–160, 1996.

[Fet03] Christof Fetzer. Perfect failure detection in timed asynchronous systems. *IEEE Trans. Comput.*, 52(2):99–112, 2003.

[FLP85] M. J. Fischer, N. Lynch, and M. S. Peterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FMR05] Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Trans. Dependable Secur. Comput.*, 2(1):46–56, 2005.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977.

[GDG+03] R. Guimerà, L. Danon, Díaz A. Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103+, Dec 2003.

[GGN09] Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Of malicious motes and suspicious sensors. *Theor. Comput. Sci.*, 410(6-7):546–569, 2009.

[GKMM07] Ayalvadi Ganesh, Anne-Marie Kermarrec, Erwan Le Merrer, and Laurent Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4):267–278, 2007.

[GL08] Natali Gulbahce and Sune Lehmann. The art of community detection. Jul 2008.

[GSS08] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2008.

[Gue00] Rachid Guerraoui. Indulgent algorithms (preliminary version). In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 289–297, New York, NY, USA, 2000. ACM.

[Gus03] F. Gustafsson. Positioning using time-difference of arrival measurements. In *Proceedings of ICASSP*, 2003.

[Has70] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[HC07] Elyes Ben Hamida and Guillaume Chelius. Analytical evaluation of virtual infrastructures for data dissemination in wireless sensor networks with mobile sink. In *SANET '07: Proceedings of the First ACM workshop on Sensor and actor networks*, pages 3–10, New York, NY, USA, 2007. ACM.

[HDYK04] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. The φ accrual failure detector. In *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*, volume 0, pages 66–78, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[HH95] Per Hage and Frank Harary. Eccentricity and centrality in networks. *Social Networks*, 17(1):57 – 63, 1995.

[HHB+03] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-free localization schemes for large scale sensor networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 81–95, New York, NY, USA, 2003. ACM.

[HMSZ55] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Chasing the weakest system model for implementing omega and consensus. *IEEE Transactions on Dependable and Secure Computing*, 99(1), 5555.

[HMT07] Moumen Hamouma, Achour Mostéfaoui, and Gilles Trédan. Byzantine consensus with few synchronous links. In *OPODIS*, pages 76–89, 2007.

[HWLC01] B. Holfmann-Wellnhof, H. Lichtenegger, and J. Collins. *Global Positioning Systems: Theory and Practice*. Springer, 5th edition, 2001.

[JGKvS04] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. *Lecture Notes in Computer Science*, 3231:79–98, 2004.

[Joh03] C.W.; Johnson. *Failure in Safety Critical Systems: A Handbook of Accident and Incident Reporting*. University of Glasgow, 2003.

[Kle00] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, New York, NY, USA, 2000. ACM.

[KLMST09] A.-M. Kermarrec, E. Le Merrer, B. Sericola, and G. Trédan. Rr-6809 inria - second order centrality:distributed assessment of nodes criticity in complex networks, 2009.

[KMMS97] Kim Potter Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Solving consensus in a byzantine environment using an unreliable fault detector. In *In Proceedings of the International Conference on Principles of Distributed Systems (OPODIS*, pages 61–75, 1997.

[KMR+07] Anne-Marie Kermarrec, Achour Mostefaoui, Michel Raynal, Gilles Trédan, and Aline Viana. Large-scale networked systems: from anarchy to geometric self-structuring. Research report, Irisa, 2007.

[KMR+09] Anne-Marie Kermarrec, Achour Mostéfaoui, Michel Raynal, Gilles Trédan, and Aline Carneiro Viana. Large-scale networked systems: From anarchy to geometric self-structuring. In *ICDCN*, pages 25–36, 2009.

[KMST08] Anne-Marie Kermarrec, Erwan Le Merrer, Bruno Séricola, and Gilles Trédan. Evaluating the quality of a network topology through random walks. In *DISC 2008, International Symposium on Distributed Computing*, September 2008.

[KSC03] Sudha Krishnamurthy, William H. Sanders, and Michel Cukier. An adaptive quality of service aware middleware for replicated services. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1112–1125, 2003.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lam06] Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, October 2006.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[LEA+01] Fredrik Liljeros, Christofer R. Edling, Luis A. Nunes Amaral, H. Eugene Stanley, and Yvonne Aberg. The web of human sexual contacts. *Nature*, 411:907, 2001.

[LM06] Matthieu Latapy and Clémence Magnien. Measuring fundamental properties of real-world complex networks. *CoRR*, abs/cs/0609115, 2006.

[LMT09] Erwan Le Merrer and Gilles Trédan. Centralities: capturing the fuzzy notion of importance in social graphs. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 33–38, New York, NY, USA, 2009. ACM.

[Lov98] L. Lovàsz. Random walks on graphs: A survey,. In *Combinatorics, Paul Erdos is Eighty, Vol. 2 (ed. D. Miklós, V. T. Sós, T. Szonyi)*, 1998.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[MA06] Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3):202–215, 2006.

[MMR03] Achour Mostefaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. pages 351–360. DSN, 2003.

[Moh97] B. Mohar. Some applications of laplace eigenvalues of graphs, 1997.

[MOWW04] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proc. of Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, October 2004.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[MRT09] Achour Mostéfaoui, Michel Raynal, and Gilles Trédan. On the fly estimation of the processes that are alive in an asynchronous message-passing system. *IEEE Trans. Parallel Distrib. Syst.*, 20(6):778–787, 2009.

[New05] Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, January 2005.

[New06] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74:036104, 2006.

[NSB03] Radhika Nagpal, Howard E. Shrobe, and Jonathan Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *IPSN*, pages 333–348, 2003.

[OW05] R. O'Dell and R. Wattenhofer. Theoretical aspects of connectivity-based multi-hop positioning. *Theor. Comput. Sci.*, 344(1):47–68, 2005.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[PDFV05] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, June 2005.

[pee] http://peersim.sourceforge.net.

[Pow92] David Powell. Failure mode assumptions and assumption coverage. In *FTCS*, pages 386–395, 1992.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *SFCS '83: Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 403–409, Washington, DC, USA, 1983. IEEE Computer Society.

[Ray05] Michel Raynal. A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News*, 36(1):53–70, 2005.

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht, 2004.

[RMH98] Robbert Van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. pages 55–70, 1998.

[RRP+03] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proc. of ACM Mobicom*, April 2003.

[SOS+08] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. Nanoecc: Testing the limits of elliptic curve cryptography in sensor networks. *European conference on Wireless Sensor Networks (EWSN'08)*, 2008.

[SRD+06] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. On unbiased sampling for unstructured peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 27–40, New York, NY, USA, 2006. ACM.

[SRZF03] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. Localization from mere connectivity. In *Proc. of ACM Mobihoc*, June 2003.

[SS83] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems, 1983.

[ST84] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. 1984.

[TP07] Patrick Tague and Radha Poovendran. Modeling adaptive node capture attacks in multi-hop wireless networks. *Ad Hoc Networks*, 5(6):801–814, 2007.

[VGS05] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13:2005, 2005.

[vid] http://www.irisa.fr/asap/intranet/second_order.avi.

[ZGG05] Hui Zhang, Ashish Goel, and Ramesh Govindan. An empirical evaluation of internet latency expansion. *SIGCOMM Comput. Commun. Rev.*, 35(1):93–97, 2005.

[ZXSJ03] Sencun Zhu, Shouhuai Xu, Sanjeev Setia, and Sushil Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: A probabilistic approach. In *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*, page 326, Washington, DC, USA, 2003. IEEE Computer Society.

[ZZF09] Chi Zhang, Yanchao Zhang, and Yuguang Fang. Localized algorithms for coverage boundary detection in wireless sensor networks. *Wirel. Netw.*, 15(1):3–20, 2009.